owner='S08:WK:HFI'

```
 SSSSS     00       888           W     W  K     K          H     H
S     S   0  0     8   8          W     W  K    K           H     H
S         0    0   8     8   **    W     W  K   K      **    H     H
S         0    0   8   8     **    W     W  K  K       **    H     H
 SSSSS    0    0    88888         W  W  W  K K              HHHHHHH
      S   0    0   8     8         W  W  W  KKK             H     H
      S   0    0   8     8   **    W W W W  K  K       **    H     H
      S   0    0   8     8   **    WW   WW  K   K      **    H     H
S     S    0  0    8     8         W     W  K    K          H     H
 SSSSS      00     88888           W     W  K     K         H     H
```

Despooler of 3Share - version 1.3.1

file spooled: 4-jul-89, 11:09:10
file printed: 4-jul-89, 11:33:01
copies=1, priority=50, form type=1

1 0

```
 0
 1 Video display interface for an ANSI.SYS interface.
 2 It should work on any MS-DOS computer. Since ANSI.SYS does
 3 not have a delete line function, split screen can not
 4 be implemented as usual. Instead, the cursor "rotates"
 5 ie. when a CR is performed on the bottom line, the cursor
 6 moves up to the top line in the current window.
 7
 8
 9
10
11
12
13
14
15
```

1 1

```
 0 \ ansi cursor steurung                          ks 31 aug 86
 1   Onlyforth
 2
 3 ¦ : (char"    "lit count bounds DO I c@ charout LOOP ;
 4 ¦ : char"     compile (char" ," align ; immediate restrict
 5
 6 ¦ Ascii 0 Constant #0
 7
 8 ¦ : (#S)  ( u -- )  &10 /mod  #0 + charout   #0 + charout ;
 9
10   : (at  ( row col -- )     char" [["
11      swap 1+ (#S) char" ;" 1+ (#S) char" H" ;
12
13 ¦ : )##(  ( -- u )   (key #0 - &10 *   (key #0 -  + ;
14                        .
15   1 4 +thru   .( ANSI display interface active) cr
```

1 2

```
 0 \ Ansii Standard display output                 ks  1 sep 86
 1 ¦ : keydrop    (key drop ;
 2
 3   : (at?    char" [[6n"    keydrop   keydrop
 4      )##( 1-  keydrop  )##( 1-  keydrop keydrop ;
 5
 6   Variable top   top off
 7
 8   : full         top off ;
 9
10   : blankline    char" [[K" ;
11 ¦ : lineerase    0 (at blankline ;
12
13   : normal     char" [[0m" ;  : invers     char" [[7m" ;
14   : underline  char" [[4m" ;  : bright     char" [[1m" ;
15
```

3

```
\ Ansii Standard display output                ks  1 sep 86

  ' 2drop Alias curshape
  ' drop  Alias setpage
  ' (at?  Alias curat?

  : (type  ( addr len -- )   pad place
     pad count bounds ?DO  I c@ (emit  LOOP ;

  : (cr   top @ 0=  adr .status @ ['] noop = and
     IF  (cr exit  THEN   row c/col 2- u<
     IF  row 1+  ELSE  top @  THEN  lineerase ;

  : (page   top @ 0= IF  char" [[2J" exit  THEN
     top @ c/col 2- DO  I lineerase  -1 +LOOP ;
```

4

```
\ statuszeile                                  ks  1 sep 86

  ' (cr   ' display   4 + !  ' (type ' display   6 + !
  ' (page ' display &10 + !
  ' (at   ' display &12 + !  ' (at?  ' display &14 + !


  : .sp      ( n -- )  ." s" depth   swap 1+ - 2 .r ;
  : .base    base @  decimal dup 2 .r   base ! ;
  : (.drv    ( n -- )   Ascii A + emit ." : " ;
  : .dr      ."    " drv (.drv ;
  : .scr     blk @ IF  ."   Blk" blk  ELSE  ."   Scr" scr  THEN
             @ 5 .r ;
  : .space   ."   Dic" s0 @ here $100 + - 6 u.r ;
```

5

```
\ statuszeile                                  ks  1 sep 86

¦ : fstat  ( n -- )   invers .base .sp
     .space .scr .dr file? 2 spaces order normal ;

  : .stat    output @  (at?   display  c/col 1- 0 (at
     3 fstat blankline   (at  output ! ;

  : +stat   ['] .stat Is .status  .status ;

  : -stat   ['] noop Is .status ;
```

1 0

```
 0 \ 8086 Assembler                                  ks 11 mai 88
 1 Der 8086 Assembler wurde von Klaus Schleisiek geschrieben.
 2 Assembler Definitionen werden durch das definierende Wort
 3 CODE erzeugt und durch END-CODE abgeschlossen.
 4
 5 Die Register des 8086 werden im volksFORTH folgendermaßen
 6 benutzt und benannt:
 7 Intel  Forth  Benutzt für              8bit-Register
 8 AX      A     frei                        A+ A-
 9 DX      D     oberstes Stackelement       D+ D-
10 CX      C     frei                        C+ C-
11 BX      R     Returnstack Pointer         R+ R-
12 BP      U     User Pointer
13 SP      S     Stack Pointer
14 SI      I     Instruction Pointer
15 DI      W     Word Pointer, meist frei benutzbar
```

1 1

```
 0 \ 8086 Assembler   loadscreen                  ks 19 jun 88
 1   Onlyforth
 2
 3 ¦ : u2/ ( 16b -- 15b )   2/ $7FFF and ;
 4 ¦ : 8*  ( 15b -- 16b )   2* 2* 2* ;
 5 ¦ : 8/  ( 16b -- 13b )   u2/ 2/ 2/ ;
 6
 7   Vocabulary Assembler
 8   Assembler also definitions
 9
10   3 &21 thru clear  .( Assembler geladen) cr
11
12
13
14
15
```

1 2

```
 0 \ conditional Assembler compiler               ks 11 jul 86
 1   here
 2
 3   : temp-assembler  ( addr -- )  hide  last off  dp !
 4      " ASSEMBLER"  find nip ?exit   here $1800 + sp@ u>
 5      IF  display cr ." Assembler passt nicht" abort  THEN
 6      here   sp@ $1800 - dp !  1 load   dp ! ;
 7
 8   temp-assembler   \\
 9
10   : blocks   ( n -- addr / ff )
11      first @ >r   dup 0 ?DO  freebuffer  LOOP
12      [ b/blk negate ] Literal * first @ +   r@ u> r> and ;
13
14
15
```

1 11

```
 \ 8086 Arithmetic instructions                   ks 25 mai 87

 ¦ : Arith: ( code -- )  Create ,
   Does> @ >r   2address  immediate?
      IF  rmode? IF  ?akku IF  r> size @
                               IF  5 or >c, >, wexit  THEN
                               4 or >c, >c, wexit  THEN THEN
          r@ or  $80 size @ or   r> 0<
          IF  size @ IF  2 pick long? 0= IF  2 or  size off  THE
          THEN      THEN  >c, >c, direct,  data,  wexit
      THEN  r> dw, r/m,  wexit ;

   $8000 Arith: add      $0008 Arith: or
   $8010 Arith: adc      $8018 Arith: sbb
   $0020 Arith: and      $8028 Arith: sub
   $0030 Arith: xor      $8038 Arith: cmp
```

1 12

```
 \ 8086 move push pop                             ks 25 mai 87

   : mov    [ Forth ] 2address  immediate?
      IF  rmode? IF  r/m $80 or size @ IF  8 or  THEN
                     >c, data, wexit
                 THEN  $C6 w, r/m, data, wexit
      THEN  6 case? IF  $A2 dw, direct, wexit  THEN
      smode? IF  $8C direction @ IF  2 or  THEN  >c, r/m, wexit
             THEN  $88 dw, r/m, wexit ;

   ¦ : pupo   [ Forth ] >r  1address  ?word
      smode? IF  reg 6 r> IF  1+  THEN  or >c, wexit  THEN
      rmode? IF  r/m $50 or r> or >c, wexit  THEN
      r> IF  $8F  ELSE  $30 or $FF  THEN  >c, r/m, wexit ;

   : push  0 pupo ;        : pop   8 pupo ;
```

1 13

```
 \ 8086 inc & dec , effective addresses           ks 25 mai 87

 ¦ : inc/dec   [ Forth ] >r 1address   rmode?
      IF  size @ IF  r/m $40 or r> or >c, wexit  THEN
      THEN  $FE w, r> or r/m, wexit ;

   : dec   8 inc/dec ;         : inc   0 inc/dec ;

 ¦ : EA:  ( code -- )   Create c, [ Forth ]
   Does> >r 2address nonimmediate
      rmode? direction @ 0= or ?moderr r> c@ >c, r/m, wexit ;

   $C4 EA: les   $8D EA: lea   $C5 EA: lds
```

1 3

```
0 \ Code generating primitives                    ks  5 nov 86
1
2  Variable >codes  \ points at table of execution vectors
3
4 ¦ Create nrc  ] c, , here ! c! [
5
6   : nonrelocate  nrc >codes ! ;  nonrelocate
7
8 ¦ : >exec ( n -- n+2 )   Create dup c, 2+
9     Does> c@  >codes @ + perform ;
10
11 0 ¦ >exec >c,       ¦ >exec >,       ¦ >exec >here
12   ¦ >exec >!        ¦ >exec >c!   drop
13
14
15
```

1 4

```
0 \ 8086 Registers                                ks 25 mai 87
1
2  0 Constant A    1 Constant C    2 Constant D    3 Constant R
3  4 Constant S    5 Constant U    6 Constant I    7 Constant W
4 ' I Alias SI    ' W Alias DI    ' R Alias BX
5
6  8 Constant A-   9 Constant C-  $A Constant D-  $B Constant R-
7 $C Constant A+  $D Constant C+  $E Constant D+  $F Constant R+
8 ' R- Alias B-   ' R+ Alias B+
9
10  $100 Constant E:         $101 Constant C:
11  $102 Constant S:         $103 Constant D:
12
13 ¦ Variable isize        ( specifies Size by prefix)
14 ¦ : Size: ( n -- )  Create c,  Does>  c@ isize ! ;
15  0 Size: byte      1 Size: word  word    2 Size: far
```

1 5

```
0 \ 8086 Assembler  System variables              ks 25 mai 87
1
2 ¦ Variable direction    \ 0 reg>EA, -1 EA>reg
3 ¦ Variable size         \ 1 word, 0 byte, -1 undefined
4 ¦ Variable displaced    \ 1 direct, 0 nothing, -1 displaced
5 ¦ Variable displacement
6
7 ¦ : setsize             isize @  size ! ;
8 ¦ : long?   ( n -- f )  $FF80 and dup 0< not ?exit $FF80 xor ;
9 ¦ : wexit               rdrop word ;
10 ¦ : moderr             word true Abort" invalid" ;
11 ¦ : ?moderr ( f -- )   0=exit  moderr ;
12 ¦ : ?word              size @ 1- ?moderr ;
13 ¦ : far?   ( -- f )    size @ 2 = ;
14
15
```

1 14

```
\ 8086  xchg  segment prefix                      ks 25 mai 87
  : xchg  [ Forth ]  2address nonimmediate rmode?
   IF  size @ IF  dup r/m 0=
                  IF  8/ true  ELSE  dup $38 and 0=  THEN
                  IF  r/m $90 or >c, wexit  THEN
   THEN       THEN  $86 w, r/m, wexit ;

¦ : 1addr:  ( code -- )  Create c, [ Forth ]
  Does> c@ >r 1address $F6 w, r> or r/m, wexit ;

  $10 1addr: com     $18 1addr: neg
  $20 1addr: mul     $28 1addr: imul
  $38 1addr: idiv    $30 1addr: div

  : seg   ( 8b -)    [ Forth ]
    $100 xor dup $FFFC and ?moderr  8* $26 or >c, ;
```

1 15

```
\ 8086  test not neg mul imul div idiv            ks 26 mai 87

  : test   [ Forth ]  2address immediate?
    IF  rmode? IF  ?akku IF  $A8 w, data, wexit  THEN THEN
        $F6 w, r/m, data, wexit
    THEN  $84 w, r/m, wexit ;

¦ : in/out  [ Forth ] >r 1address setsize
    $C2 case? IF  $EC r> or w, wexit  THEN
    6 - ?moderr  $E4 r> or w,  displacement @ >c, wexit ;

  : out  2 in/out ;          : in   0 in/out ;

  : int    3 case? IF  $CC >c, wexit  THEN  $CD >c, >c, wexit ;
```

1 16

```
\ 8086 shifts  and  string instructions           ks 26 mai 87

¦ : Shifts:  ( code -- )   Create c,  [ Forth ]
  Does> c@ >r C* case? >r 1address
        r> direction !  $D0 dw, r> or r/m, wexit ;

  $00 Shifts: rol     $08 Shifts: ror
  $10 Shifts: rcl     $18 Shifts: rcr
  $20 Shifts: shl     $28 Shifts: shr
  $38 Shifts: sar      ' shl Alias sal

¦ : Str:  ( code -- )   Create c,
  Does> c@ setsize w, wexit ;

  $A6 Str: cmps      $AC Str: lods     $A4 Str: movs
  $AE Str: scas      $AA Str: stos
```

1 6

```
 0 \ 8086 addressing modes                          ks 25 mai 87
 1
 2 ¦ Create (EA  7 c, 0 c, 6 c, 4 c, 5 c,
 3 ¦ : ()  ( 8b1 -- 8b2 )
 4     3 - dup 4 u> over 1 = or ?moderr (EA + c@ ;
 5
 6  -1 Constant #       $C6 Constant #)       -1 Constant C*
 7
 8   : )   ( u1 -- u2 )
 9     () 6 case? IF  0 $86 exit  THEN  $C0 or ;
10   : I)  ( u1 u2 -- u3 )  + 9 - dup 3 u> ?moderr $C0 or ;
11
12   : D)  ( n u1 -- n u2 )
13     () over long? IF  $40  ELSE  $80  THEN or ;
14   : DI) ( n u1 u2 -- n u3 )
15     I) over long? IF  $80  ELSE  $40  THEN xor ;
```

1 7

```
 0 \ 8086 Registers and addressing modes            ks 25 mai 87
 1
 2 ¦ : displaced?  ( [n] u1 -- [n] u1 f )
 3     dup #) = IF  1 exit  THEN
 4     dup $C0 and dup $40 = swap $80 = or ;
 5
 6 ¦ : displace    ( [n] u1 -- u1 )   displaced? ?dup 0=exit
 7     displaced @ ?moderr   displaced !   swap displacement !  ;
 8
 9 ¦ : rmode   ( u1 -- u2 )   1 size !  dup 8 and 0=exit
10     size off  $FF07 and ;
11
12 ¦ : mmode?  ( 9b - 9b f)     dup $C0 and ;
13
14 ¦ : rmode?  ( 8b1 - 8b1 f)   mmode? $C0 = ;
15
```

1 8

```
 0 \ 8086  decoding addressing modes                ks 25 mai 87
 1
 2 ¦ : 2address  ( [n] source [displ] dest -- 15b / [n] 16b )
 3     size on   displaced off   dup # = ?moderr   mmode?
 4     IF  displace False  ELSE  rmode True  THEN  direction !
 5     >r # case?  IF  r> $80C0 xor  size @ 1+ ?exit  setsize exit
 6                 THEN  direction @
 7     IF  r> 8* >r mmode? IF  displace
 8         ELSE  dup 8/ 1 and  size @ = ?moderr $FF07 and  THEN
 9     ELSE  rmode 8*
10     THEN  r> or $C0 xor ;
11
12 ¦ : 1address  ( [displ] 9b -- 9b )
13     # case? ?moderr   size on   displaced off   direction off
14     mmode? IF  displace setsize  ELSE  rmode  THEN  $C0 xor ;
15
```

1 17

```
 \ implied 8086 instructions                        ks 26 mai 87

   : Byte:  ( code -- )  Create c,  Does> c@ >c, ;
   : Word:  ( code -- )  Create ,   Does> @ >, ;

  $37 Byte: aaa    $AD5 Word: aad    $AD4 Word: aam
  $3F Byte: aas     $98 Byte: cbw     $F8 Byte: clc
  $FC Byte: cld     $FA Byte: cli     $F5 Byte: cmc
  $99 Byte: cwd     $27 Byte: daa     $2F Byte: das
  $F4 Byte: hlt     $CE Byte: into    $CF Byte: iret
  $9F Byte: lahf    $F0 Byte: lock    $90 Byte: nop
  $9D Byte: popf    $9C Byte: pushf   $9E Byte: sahf
  $F9 Byte: stc     $FD Byte: std     $FB Byte: sti
  $9B Byte: wait    $D7 Byte: xlat
  $C3 Byte: ret     $CB Byte: lret
  $F2 Byte: rep     $F2 Byte: 0<>rep  $F3 Byte: 0=rep
```

1 18

```
 \ 8086  jmp call  conditions                       ks 26 mai 87
 ¦ : jmp/call    >r setsize # case?  [ Forth ]
     IF  far? IF  r> IF $EA ELSE $9A THEN  >c, swap >, >, wexit
              THEN  >here 2+ - r>
         IF  dup long? 0= IF  $EB >c, >c, wexit  THEN  $E9
         ELSE  $E8  THEN  >c, 1- >, wexit
     THEN  1address $FF >c, $10 or r> +
     far? IF  8 or  THEN  r/m, wexit ;
   : call    0 jmp/call ;          : jmp   $10 jmp/call ;

  $71 Constant OS    $73 Constant CS
  $75 Constant 0=    $77 Constant >=
  $79 Constant 0<    $7B Constant PE
  $7D Constant <     $7F Constant <=
  $E2 Constant C0=   $E0 Constant ?C0=
  : not  1 [ Forth ] xor ;
```

1 19

```
 \ 8086 conditional branching                       ks 27 mai 87

   : +ret   $C2 >c, >, ;
   : +lret  $CA >c, >, ;
 ¦ : ?range    dup long? abort" out of range" ;
   : ?[     >, >here 1- ;
   : ]?     >here over 1+ - ?range swap >c! ;
   : ][     $EB ?[ swap ]? ;
   : ?[[    ?[ swap ;
   : [[     >here ;
   : ?]     >c, >here 1+ - ?range >c, ;
   : ]]     $EB ?] ;
   : ]]?    ]] ]? ;
```

1 9

```
 0 \ 8086 assembler                                  ks 25 mai 87
 1 ¦ : immediate?   ( u -- u f )  dup 0< ;
 2
 3 ¦ : nonimmediate ( u -- u )    immediate? ?moderr ;
 4
 5 ¦ : r/m                        7 and ;
 6
 7 ¦ : reg                        $38 and ;
 8
 9 ¦ : ?akku  ( u -- u ff / tf )  dup r/m 0= dup 0=exit  nip ;
10
11 ¦ : smode? ( u1 -- u1 ff / u2 tf )  dup $F00 and
12     IF  dup $100 and IF  dup r/m 8* swap reg 8/
13                          or $C0 or  direction off
14                      THEN  True exit
15     THEN  False ;
```

1 10

```
 0 \ 8086 Registers and addressing modes             ks 25 mai 87
 1
 2 ¦ : w,          size @ or >c, ;
 3
 4 ¦ : dw,         size @ or  direction @ IF  2 xor  THEN  >c, ;
 5
 6 ¦ : ?word,  ( u1 f -- )  IF  >, exit  THEN  >c, ;
 7
 8 ¦ : direct,     displaced @ 0=exit
 9     displacement @ dup long?  displaced @ 1+ or ?word, ;
10
11 ¦ : r/m,       >c, direct, ;
12
13 ¦ : data,      size @ ?word, ;
14
15
```

1 0

```
 0 \ 8086 Assembler                                  ks 11 mai 88
 1 Der 8086 Assembler wurde von Klaus Schleisiek geschrieben.
 2 Assembler Definitionen werden durch das definierende Wort
 3 CODE erzeugt und durch END-CODE abgeschlossen.
 4
 5 Die Register des 8086 werden im volksFORTH folgendermaßen
 6 benutzt und benannt:
 7 Intel  Forth  Benutzt für                 8bit-Register
 8 AX       A    frei                           A+ A-
 9 DX       D    oberstes Stackelement          D+ D-
10 CX       C    frei                           C+ C-
11 BX       R    Returnstack Pointer            R+ R-
12 BP       U    User Pointer
13 SP       S    Stack Pointer
14 SI       I    Instruction Pointer
15 DI       W    Word Pointer, meist frei benutzbar
```

20

```
\ Next  user'  end-code  ;c:                        ks 11 mär 89

  : Next  lods  A W xchg  W ) jmp
    >here   next-link @ >,   next-link ! ;

  : u'   ' >body c@ ;

  Forth definitions

\needs end-code    : end-code   toss also ;

  Assembler definitions

  : ;c:   recover # call last off  end-code  0 ] ;
```

21

```
\ 8086 Assembler, Forth words                       ks 11 mär 89
  Onlyforth

  : Assembler     Assembler [ Assembler ] wexit ;

  : ;code   0 ?pairs compile (;code
    reveal  [compile] [  Assembler ; immediate

  : Code    Create [ Assembler ] >here dup 2- >!  Assembler ;

  : >label   ( addr -- )
    here  ¦ Create immediate swap ,  4 hallot
    here 4 - heap 4 cmove   heap  last @ (name> !   dp !
  Does>  ( -- addr )  @  state @ 0=exit  [compile] Literal ;

  : Label   [ Assembler ] >here >label  Assembler ;
```

0


```
\ 8086 Assembler                                    ks 11 mai 88
Der 8086 Assembler wurde von Klaus Schleisiek geschrieben.
Assembler Definitionen werden durch das definierende Wort
CODE erzeugt und durch END-CODE abgeschlossen.

Die Register des 8086 werden im volksFORTH folgendermaßen
benutzt und benannt:
Intel  Forth  Benutzt für                 8bit-Register
AX       A    frei                           A+ A-
DX       D    oberstes Stackelement          D+ D-
CX       C    frei                           C+ C-
BX       R    Returnstack Pointer            R+ R-
BP       U    User Pointer
SP       S    Stack Pointer
SI       I    Instruction Pointer
DI       W    Word Pointer, meist frei benutzbar
```

1 0

```
 0
 1 This video display interface utilizes the ROM BIOS call $10.
 2 The display is fairly fast and should work on most IBM
 3 compatible computers
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

1 1

```
 0 \ BIOS display interface                    ks  1 sep 86
 1   Onlyforth   \needs Assembler   2 loadfrom asm.scr
 2   Variable dpage        dpage off
 3   Variable top          top off
 4
 5   Code (at    ( lin col -- )   A pop   R push   U push
 6      dpage #) R+ mov   A- D+ mov   2 # A+ mov   $10 int
 7      U pop   R pop   D pop   Next   end-code
 8
 9   Code (at?   ( -- lin col )   D push   R push   U push
10      dpage #) R+ mov   3 # A+ mov   $10 int   U pop   R pop
11      D+ A- mov   0 # A+ mov   A+ D+ mov   A push   Next
12   end-code
13
14   1 6 +thru   .( BIOS display interface active) cr
15
```

1 2

```
 0 \ BIOS  normal invers  blankline              ks  1 sep 86
 1   : full           top off ;
 2
 3   Variable attribut     7 attribut !
 4
 5   : normal     7 attribut ! ;   : invers   $70 attribut ! ;
 6   : underline  1 attribut ! ;   : bright    $F attribut ! ;
 7
 8   Code blankline      D push   R push   U push
 9      dpage #) R+ mov   attribut #) R- mov
10      3 # A+ mov   $10 int   ' c/row >body #) C mov
11      D- C- sub   bl # A- mov   9 # A+ mov   $10 int
12      U pop   R pop   D pop   Next   end-code
13
14 ¦ : lineerase   0 (at blankline ;
15
```

4

```
\ BIOS  (type  (emit                            ks  1 sep 86

  Code (type   ( addr len -- )   W pop   R push   U push
     D U mov   dpage #) R+ mov   attribut #) R- mov
     3 # A+ mov   $10 int   U inc   C push   $E0E # C mov
     1 # A+ mov   $10 int   1 # C mov   [[  U dec  0= not
     ?[[  D- inc   ' c/row >body #) D- cmp  0= not
          ?[[  W ) A- mov   W inc   9 # A+ mov
               $10 int   2 # A+ mov   $10 int  ]]?
     ]?   C pop   1 # A+ mov   $10 int
     U pop   R pop   D pop   ' pause #) jmp
  end-code

  : (emit  ( char -- )   sp@ 1 (type drop ;
```

5

```
\ BIOS  (del scroll (cr (page                   ks  2 sep 86

  : (del    (at? ?dup
     IF  1- 2dup (at  bl (emit  (at exit  THEN  drop ;

  Code scroll   D push   R push   U push   attribut #) R+ mov
     top #) C+ mov   0 # C- mov   ' c/row >body #) D- mov
     D- dec   ' c/col >body #) D+ mov   D+ dec   D+ dec
     $601 # A mov   $10 int   U pop   R pop   D pop   Next
  end-code

  : (cr     (at? drop 1+ dup 2+ c/col u>
     IF  scroll 1-  THEN  lineerase ;

  : (page   top @ c/col 2- 0DO  I lineerase  -1 +LOOP ;
```

6

```
\ BIOS  status display                          ks  2 sep 86

  ' (emit ' display   2 + !   ' (cr   ' display   4 + !
  ' (type ' display   6 + !   ' (del  ' display   8 + !
  ' (page ' display &10 + !
  ' (at   ' display &12 + !   ' (at?  ' display &14 + !

  : .sp      ( n -- )  ." s" depth   swap 1+ - 2 .r ;
  : .base    base @  decimal dup 2 .r   base ! ;
  : (.drv    ( n -- )   Ascii A + emit ." : " ;
  : .dr      ."    " drv (.drv ;
  : .scr     blk @ IF  ."   Blk" blk  ELSE  ."   Scr" scr  THEN
             @ 5 .r ;
  : .space   ."   Dic" s0 @ here $100 + - 6 u.r ;
```

1 3

```
0 \ curshape  setpage  curat?                    ks  8 mar 88
1
2  Code curshape  ( top bot -- )   D C mov   D pop
3     D- C+ mov   1 # A+ mov   $10 int   D pop   Next
4  end-code
5
6  Code setpage   ( n -- )
7     $503 # A mov   D- A- and   $10 int   D pop   Next
8  end-code
9
10 ' (at? Alias curat?
11
12
13
14
15
```

1 O

```
0
1 This video display interface utilizes the ROM BIOS call $10.
2 The display is fairly fast and should work on most IBM
3 compatible computers
4
5
6
7
8
9
10
11
12
13
14
15
```

1 O

0
1 This video display interface utilizes the ROM BIOS call $10.
2 The display is fairly fast and should work on most IBM
3 compatible computers
4
5
6
7
8
9
10
11
12
13
14
15

1 7

```
\ statuszeile                                    ks  1 sep 86

: fstat  ( n -- )   .base .sp
    .space .scr .dr file? 2 spaces order ;

 : .stat    attribut @   output @  (at?
    display  invers  c/col 1- 0 (at  4 fstat
    blankline  (at   output !   attribut ! ;

 : +stat   ['] .stat Is .status  .status ;

 : -stat   ['] noop Is .status ;
```

O

This video display interface utilizes the ROM BIOS call $10.
The display is fairly fast and should work on most IBM
compatible computers

1 O

```
\                                                        ks 11 mai 88
Zum Kopieren von physikalischen Blöcken in Files hinein.

Der Kopiervorgang findet statt vom aktuellen File und Lauf-
werk in ein neues File auf dem Laufwerk und in dem Sub-
directory, das gerade für MS-DOS Files aktuell ist, d.h. im
DIRECT-Modus kann ein anderes Laufwerk gewählt sein als im
FILE-Modus.

Mit folgender Sequenz werden die physikalischen Blöcke
10 - 20 auf Laufwerk C: in das File TEST.SCR im Subdirectory
D:\VOLKS kopiert

KERNEL.SCR D: CD \VOLKS
DIRECT C:
10 20 BLOCKS>FILE TEST.SCR
```

1

```
\ absolute blocks in file übertragen                    ks 11 mai 88

| File outfile

  : blocks>file   ( <filename> from to -- )  [ Dos ]
     isfile@ -rot   outfile make  1+ swap
     ?DO  I over (block
         ds@ swap b/blk isfile@ lfputs
     LOOP  close isfile ! ;
```

1 O

```
\                                                        ks 11 mai 88
Zum Kopieren von physikalischen Blöcken in Files hinein.

Der Kopiervorgang findet statt vom aktuellen File und Lauf-
werk in ein neues File auf dem Laufwerk und in dem Sub-
directory, das gerade für MS-DOS Files aktuell ist, d.h. im
DIRECT-Modus kann ein anderes Laufwerk gewählt sein als im
FILE-Modus.

Mit folgender Sequenz werden die physikalischen Blöcke
10 - 20 auf Laufwerk C: in das File TEST.SCR im Subdirectory
D:\VOLKS kopiert

KERNEL.SCR D: CD \VOLKS
DIRECT C:
10 20 BLOCKS>FILE TEST.SCR
```

O

```
\                                                        ks 11 mai 88
Zum Kopieren von physikalischen Blöcken in Files hinein.

Der Kopiervorgang findet statt vom aktuellen File und Lauf-
werk in ein neues File auf dem Laufwerk und in dem Sub-
directory, das gerade für MS-DOS Files aktuell ist, d.h. im
DIRECT-Modus kann ein anderes Laufwerk gewählt sein als im
FILE-Modus.

Mit folgender Sequenz werden die physikalischen Blöcke
10 - 20 auf Laufwerk C: in das File TEST.SCR im Subdirectory
D:\VOLKS kopiert

KERNEL.SCR D: CD \VOLKS
DIRECT C:
10 20 BLOCKS>FILE TEST.SCR
```

1 O

```
\                                                        ks 11 mai 88
Zum Kopieren von physikalischen Blöcken in Files hinein.

Der Kopiervorgang findet statt vom aktuellen File und Lauf-
werk in ein neues File auf dem Laufwerk und in dem Sub-
directory, das gerade für MS-DOS Files aktuell ist, d.h. im
DIRECT-Modus kann ein anderes Laufwerk gewählt sein als im
FILE-Modus.

Mit folgender Sequenz werden die physikalischen Blöcke
10 - 20 auf Laufwerk C: in das File TEST.SCR im Subdirectory
D:\VOLKS kopiert

KERNEL.SCR D: CD \VOLKS
DIRECT C:
10 20 BLOCKS>FILE TEST.SCR
```

O

```
\                                                        ks 11 mai 88
Zum Kopieren von physikalischen Blöcken in Files hinein.

Der Kopiervorgang findet statt vom aktuellen File und Lauf-
werk in ein neues File auf dem Laufwerk und in dem Sub-
directory, das gerade für MS-DOS Files aktuell ist, d.h. im
DIRECT-Modus kann ein anderes Laufwerk gewählt sein als im
FILE-Modus.

Mit folgender Sequenz werden die physikalischen Blöcke
10 - 20 auf Laufwerk C: in das File TEST.SCR im Subdirectory
D:\VOLKS kopiert

KERNEL.SCR D: CD \VOLKS
DIRECT C:
10 20 BLOCKS>FILE TEST.SCR
```

1 0

```
 0 \ Commandline EDitor für volksFORTH rev. 3.80        UH 05feb89
 1 Dieses File enthaelt Definitionen, die es ermöglichen die
 2 Kommandozeile zu editieren.
 3 Es gibt eine Commandline History, die es ermöglicht alte
 4 Eingaben wiederzuholen. Diese werden zyklisch auf Screen 0
 5 im File History gesichert und bleiben so auch über ein
 6 SAVESYSTEM erhalten.
 7
 8 Tasten:
 9   Cursor links/rechts                 [ ]
10   Zeichen löschen                     <del> und <-
11   Zeile löschen                       <esc>
12   Einfügen an aus                     <ins>
13   Zeile abschließen                   <enter>
14   Anfang/Ende der Zeile               <pos1> <end>
15   alte Zeilen wiederholen             X Y
```

1 1

```
 0 \ Commandline EDitor LOAD-Screen      UH 20Nov87   UH 05feb89
 1
 2
 3 : curleft ( -- )  at? 1- at ;
 4 : currite ( -- )  at? 1+ at ;
 5
 6 1 5 +thru  \ Erweiterte Eingabe
 7
 8 .( Kommandozeile Editor geladen ) cr
 9
10
11
12
13
14
15
```

1 2

```
 0 \ History -- Kommandogeschichte                    UH 04feb89
 1 makefile history  1 more
 2
 3 ¦ Variable line#       line# off
 4 ¦ Variable lastline#   lastline# off
 5
 6 ¦ : 'history ( n -- addr )  isfile push  history
 7     c/l *   b/blk /mod  block + ;
 8
 9 ¦ : @line ( n -- addr len )  'history c/l -trailing ;
10 ¦ : !history ( addr line# -- )
11       'history dup c/l blank  span @ c/l min cmove update ;
12 ¦ : @history ( addr line# -- )
13       @line rot swap dup span ! cmove ;
14
15 ¦ : +line ( n addr -- )  dup @  rot + l/s  mod  swap ! ;
```

1 4

```
 0 \ Erweiterte Eingabe                  UH 080Ct87   UH 05feb89
 1 ¦ : redisplay ( addr pos -- )
 2     at?  2swap  span @ swap /string type blankline at ;
 3
 4 ¦ : del   ( addr pos -- )  span @ 0=exit  dup >r   + dup 1+ swap
 5     span @ r> - cmove   -1 span +! ;
 6 ¦ : ins   ( addr pos1 -- )  dup >r   + dup  dup 1+
 7     span @ r> - cmove>   bl swap c!  1 span +! ;
 8
 9 ¦ : delete  ( a p1 -- a p2 )  2dup del  2dup redisplay ;
10 ¦ : back ( a p1 -- a p2 )  1- curleft delete ;
11
12 ¦ : recall ( a p1 -- a p2 ) at? rot - at  dup line# @ @history
13     dup 0 redisplay  at? span @ + at  span @ ;
14
15 ¦ : <start ( a1 p1 -- a2  p2 )  at?  rot - at  0 ;
```

1 5

```
 0 \ Tastenbelegung für Zeilen-Editor MS/DOS          ks 07 feb 89
 1
 2 : (decode ( addr pos1 key -- addr pos2 )
 3   -&77  case? IF dup  span @ < 0=exit  currite 1+   exit THEN
 4   -&75  case? IF dup          0=exit  curleft 1-   exit THEN
 5   -&82  case? IF insert @ 0=  insert !             exit THEN
 6    #bs  case? IF dup          0=exit  back         exit THEN
 7   -&83  case? IF span @ 2dup < and 0=exit delete   exit THEN
 8   -&72  case? IF -1 line# +line recall             exit THEN
 9   -&80  case? IF  1 line# +line recall             exit THEN
10    #cr  case? IF done                              exit THEN
11    #esc case? IF <start span off  2dup redisplay   exit THEN
12   -&71  case? IF <start                            exit THEN
13   -&79  case? IF at? rot -  span @ +  at  span @   exit THEN
14    dup emit >r  insert @ IF 2dup ins THEN   2dup +
15    r>  swap c!  1+  dup span @ max span ! 2dup redisplay ;
```

1 6

```
 0 \ Patch                                UH 080ct87   UH 04feb8
 1
 2 : showcur ( -- )
 3   insert @ IF &11 ELSE &6 THEN &12 curshape ;
 4
 5 : (expect ( addr len -- )  maxchars !   span off
 6     lastline# @ line# !  0
 7     BEGIN span @ maxchars @ u<
 8     WHILE key decode showcur REPEAT 2drop ;
 9
10 ' (decode ' keyboard 6 + !
11 ' (expect ' keyboard 8 + !
12
13
14
15
```

```
1                3                                7

 0 \ Ende der Eingabe                              UH 05feb89
 1
 2 ¦ Variable maxchars              ¦ Variable insert insert on
 3
 4 ¦ : -text ( a1 a2 l -- 0=equal )  bounds
 5       ?DO count I c@ - ?dup IF nip ENDLOOP exit THEN LOOP 0= ;
 6
 7 ¦ : done ( a p1 -- a p2 )  2dup
 8       at?  rot -  span @ dup maxchars !  + at space blankline
 9       line# @ @line  span @ = IF span @ -text 0=exit 2dup THEN
10       drop  lastline# @ !history  1 lastline# +line ;
11
12
13
14
15
```

```
1                0                                0

 0 \ Commandline EDitor für volksFORTH rev. 3.80        UH 05feb89
 1 Dieses File enthaelt Definitionen, die es ermöglichen die
 2 Kommandozeile zu editieren.
 3 Es gibt eine Commandline History, die es ermöglicht alte
 4 Eingaben wiederzuholen. Diese werden zyklisch auf Screen 0
 5 im File History gesichert und bleiben so auch über ein
 6 SAVESYSTEM erhalten.
 7
 8 Tasten:
 9   Cursor links/rechts                [ ]
10   Zeichen löschen                    <del> und <-
11   Zeile löschen                      <esc>
12   Einfügen an aus                    <ins>
13   Zeile abschließen                  <enter>
14   Anfang/Ende der Zeile              <pos1> <end>
15   alte Zeilen wiederholen            X Y
```


```
1                0                                0

 0 \ Commandline EDitor für volksFORTH rev. 3.80        UH 05feb89
 1 Dieses File enthaelt Definitionen, die es ermöglichen die
 2 Kommandozeile zu editieren.
 3 Es gibt eine Commandline History, die es ermöglicht alte
 4 Eingaben wiederzuholen. Diese werden zyklisch auf Screen 0
 5 im File History gesichert und bleiben so auch über ein
 6 SAVESYSTEM erhalten.
 7
 8 Tasten:
 9   Cursor links/rechts                [ ]
10   Zeichen löschen                    <del> und <-
11   Zeile löschen                      <esc>
12   Einfügen an aus                    <ins>
13   Zeile abschließen                  <enter>
14   Anfang/Ende der Zeile              <pos1> <end>
15   alte Zeilen wiederholen            X Y
```



```
1                0                                0

 0 \ Commandline EDitor für volksFORTH rev. 3.80        UH 05feb89
 1 Dieses File enthaelt Definitionen, die es ermöglichen die
 2 Kommandozeile zu editieren.
 3 Es gibt eine Commandline History, die es ermöglicht alte
 4 Eingaben wiederzuholen. Diese werden zyklisch auf Screen 0
 5 im File History gesichert und bleiben so auch über ein
 6 SAVESYSTEM erhalten.
 7
 8 Tasten:
 9   Cursor links/rechts                [ ]
10   Zeichen löschen                    <del> und <-
11   Zeile löschen                      <esc>
12   Einfügen an aus                    <ins>
13   Zeile abschließen                  <enter>
14   Anfang/Ende der Zeile              <pos1> <end>
15   alte Zeilen wiederholen            X Y
```

1 0

```
 0 \
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

1 1

```
 0 \ A disassembler for the 8086 by Charles Curley
 1 \ Fuer das volksForth-83 angepasst von B.Molte
 2
 3 ¦ : internal  1 ?head ! ;
 4 ¦ : external  ?head off ;
 5
 6 onlyFORTH forth DEFINITIONS  DECIMAL
 7
 8 VOCABULARY DISAM DISAM also DEFINITIONS
 9
10 2 capacity 1- thru
11 onlyforth
12
13 cr .( Mit  DIS <name> wird ein Wort disassembliert. )
14 cr .( ESC bricht die Ausgabe ab. )
15
```

1 2

```
 0 \
 1
 2 internal
 3
 4 : [and] and ;  \ the forth and
 5 : [or]  or ;
 6
 7 : mask  ( n maskb -- n n' ) over and ;
 8
 9 5 constant 5  \ ist so kuerzer!
10 6 constant 6
11 7 constant 7
12 8 constant 8
13
14
15
```

21

```
 \ conditional branches

 create branch-tab
  ," O  NO B  NB E  NE BE NBES  NS P  NP L  GE LE NLE"

 : .BRANCH    \ op ---  ¦ branch printed out w/ dest.
  NEXTB SEXT  CP@ + u.  ASCII J EMIT
  &15 [and] 3 * branch-tab 1+ + 3 type ;
```

22

```
 \
 \\
```

23

```
 \
 internal

 : MEDS       \ op ---  ¦ 40-7f opcodes printed out
  DUP 4 shift> 3 exec
  REGS  REGS  OOPS1 .BRANCH  STOP[

 : 80/81      \ op ---  ¦ secondary at 80 or 81
  NEXTB  ?DISP OVER 1 [and]  IF WDISP ELSE BIMM THEN  .# .MREG
  SWAP .SIZE  3 shift> 7 EXEC
   ADD OR  ADC SBB AND SUB XOR CMP   STOP[
```

```
1                    3

 0 \
 1  internal
 2
 3  : EXEC    [and] 2* R> + PERFORM ;
 4
 5  : STOP[
 6   0 ?pairs  [compile] [  reveal ;  immediate restrict
 7
 8  code shift>     \ n ct --- n' ¦ shift n right ct times
 9   D C mov  D pop  D C* shr   next end-code
10 \ : shift> 0 ?DO 2/ ( shift's artihm.!) $7FFF and LOOP ;
11
12  code SEXT    \ n --- n' ¦ sign extend lower half of n to upper
13   D A mov  cbw  A D mov  next end-code
14 \ : hsext  $FF and dup $80 and IF $FF00 or THEN  ;
15
```

```
1                    4

 0 \
 1  external
 2  VARIABLE RELOC  0 ,  ds@ 0 RELOC 2! \ keeps relocation factor
 3  internal
 4
 5  VARIABLE CP
 6  VARIABLE OPS \ operand count
 7
 8  : cp@ cp @ ;
 9  : C?  C@ . ;
10
11  : (T@)   RELOC 2@ ROT +  L@ ; \ in first word, seg in 2nd. You
12                                \ dump/dis any segment w/ any
13  : (TC@)  RELOC 2@ ROT + LC@ ; \ relocation you want by setting
14                                \ RELOC  correctly.
15  : SETSEG   RELOC 2+ ! ;
```

```
1                    5

 0 \
 1  external
 2
 3  DEFER T@    DEFER TC@
 4
 5  : HOMESEG   ds@ SETSEG ; HOMESEG
 6
 7  : SEG?      RELOC 2+ @ 4 U.r ;
 8
 9  : .seg:off  seg? ." :" cp@ 4 u.r  2 spaces ;
10
11  : MEMORY    ['] (TC@) IS TC@    ['] (T@) IS T@ ;   MEMORY
12
13
14
15
```

```
24

 \
 internal
 : 83S       \ op --- ¦ secondary at 83
  NEXTB  ?DISP BIMM .# .MREG
  SWAP .SIZE  3 shift>  7 EXEC
   ADD  OOPS0 ADC SBB oops0 SUB OOPS0 CMP   STOP[

 : 1GP       \ op --- ¦ r/m reg opcodes
  CREATE  LAST @ ,  DOES> @ >R  NEXTB  ?DISP .REG .MREG  2DROP
  R> .name ;

 external  1GP TEST  1GP XCHG  .SELF LEA  .SELF MOV  internal

: MOVRM/REG  NEXTB  ?DISP .REG .MREG  2DROP  MOV ; \ 88-89
: MOVD       NEXTB        .MREG .REG  2DROP  MOV ; \ 8A-8B
```

```
25

 \
 internal
: MOVS>M     \ op --- ¦ display instructions  8C-8E
 NEXTB  OVER $8D = IF  .MREG .REG  LEA  ELSE
   OVER $8F = IF  .MREG  [ ' POP >NAME ] LITERAL .name  ELSE
   SWAP 1 [or] SWAP  \ 16 bit moves only, folks!
   OVER 2 [and] IF  .MREG DUP .SEG  ELSE
     DUP .SEG .MREG  THEN  MOV  THEN THEN  2DROP ;


: 8MOVS      \ op --- ¦ display instructions  80-8F
 DUP 2/ 7 exec
  80/81 83S TEST XCHG MOVRM/REG  MOVD  MOVS>M  MOVS>M  STOP[
```

```
26

 \
 external
 .SELF XCHG  .SELF CBW   .SELF CWD  .SELF CALL    .SELF NOP
 .SELF WAIT  .SELF PUSHF .SELF POPF .SELF SAHF    .SELF LAHF
 internal

 : INTER      \ ---  ¦ decode interseg jmp or call
  NEXTW  4 u.r ." :" NEXTW U. ;

 : CALLINTER  \ ---  ¦ decode interseg call
  INTER  CALL ;

 : 9HIS       \ op --- ¦ 98-9F decodes
  7 exec
  CBW  CWD  CALLINTER WAIT  PUSHF  POPF SAHF LAHF  STOP[
```

1 6

```
 0 \
 1  internal
 2
 3
 4  : oops   ." ??? " ;
 5
 6  : OOPS0  oops  ;
 7  : OOPS1  oops drop ;
 8  : OOPS2  oops 2drop ;
 9
10
11
12
13
14
15
```

1 7

```
 0 \
 1
 2  : NEXTB  CP@ TC@    1 CP +! ;
 3  : NEXTW  CP@ T@     2 CP +! ;
 4
 5  : .myself \  ---  ¦ have the current word print out its name.
 6   LAST @ [COMPILE] LITERAL  COMPILE .name ;   IMMEDIATE
 7
 8
 9
10
11
12
13
14
15
```

1 8

```
 0 \
 1  internal
 2
 3  VARIABLE IM  \ 2nd operand extension flag/ct
 4
 5  : ?DISP   \ op ext --- op ext ¦ does MOD operand have a disp?
 6    DUP 6 shift>  DUP 3 =  OVER 0= [or]  0= IF  IM ! exit then
 7    0= IF  DUP 7 [and] 6 = IF 2 IM !  THEN  THEN ;
 8
 9
10  : .SELF      \  --  ¦ create a word which prints its name
11    CREATE  LAST @ ,  DOES> @ .name ; \ the ultimate in self-doc!
12
13
14
15
```

27

```
 \
  internal
  : XCHGA      \ op ---  ¦ 98-9F decodes
   dup $90 = IF drop NOP ELSE .A .16REG  XCHG THEN ;

  : 90S        \ op ---  ¦ 90-9F decodes
   DUP 3 shift> 1 exec XCHGA  9HIS  STOP[

  : MOVSs      \ op ---  ¦ A4-A5 decodes
    .SIZE ." MOVS " ;

  : CMPSs      \ op ---  ¦ A6-A7 decodes
    .SIZE ." CMPS " ;
```

28

```
 \
  internal
  : .AL/AX     \ op ---  ¦ decodes for size
   1 EXEC .A- .A STOP[

  : MOVS/ACC   \ op ---  ¦ A0-A3 decodes
   2 mask
    IF .AL/AX  WDISP ." ) " ELSE WDISP ." ) " .AL/AX  THEN MOV ;

  create ss-tab ," TESTSTOSLODSSCAS"

  : .ss-tab  3 [and] 4 * ss-tab 1+ + 4 type space ;

  : .TEST      \ op ---  ¦ A8-A9 decodes
   1 mask IF WDISP ELSE BIMM THEN .# .AL/AX  0 .ss-tab ;
```

29

```
 \
  internal
  : STOSs   ( op --- )  .SIZE  1 .ss-tab ; \  STOS
  : LODSs   ( op --- )  .SIZE  2 .ss-tab ; \  LODS
  : SCASs   ( op --- )  .SIZE  3 .ss-tab ; \  SCAS

  : AOS        \ op ---  ¦ A0-AF decodes
   DUP 2/ 7 exec
   MOVS/ACC MOVS/ACC  MOVSs  CMPSs .TEST STOSs LODSs SCASs STOP[

  : MOVS/IMM   \ op ---  ¦ B0-BF decodes
   8 mask
    IF  WDISP .# .16REG  ELSE  BIMM .# .8REG  THEN  MOV ;

  : HMEDS      \ op ---  ¦ op codes 80 - C0 displayed
   DUP 4 shift>  3 exec  8MOVS  90S AOS MOVS/IMM   STOP[
```

1 9

```
 0 \ register byte/word
 1  internal
 2
 3  create wreg-tab   ," ACDRSUIW"
 4  create breg-tab   ," A-C-D-R-A+C+D+R+"
 5
 6  : .16REG     \ r# ---  ¦ register printed out
 7    7 and wreg-tab 1+ + c@ emit space ;
 8
 9  : .8REG      \ r# ---  ¦ register printed out
10    7 and 2* breg-tab 1+ + 2 type space ;
11
12  : .A  0 .16reg ;  : .A-  0 .8reg ;
13  : .D  2 .16reg ;
14
15
```

1 10

```
 0 \ indizierte/indirekte Adressierung
 1
 2   internal
 3
 4   : ?d  DUP 6 shift> 3 [and] 1 3 uwithin ;
 5
 6   : .D)   ( disp_flag ext -- op )  \ indirekt
 7     ?d IF ." D" THEN ." ) " ;      \ mit/ohne Displacement
 8
 9   : .I)  ( disp_flag ext -- op )   \ indiziert indirekt
10    ?d IF ." D" THEN ." I) " ;      \ mit/ohne Displacement
11
12
13
14
15
```

1 11

```
 0 \ indizierte/indirekte Adressierung
 1   internal
 2
 3   : I)  6 .16reg .D) ;
 4   : W)  7 .16reg .D) ;
 5   : R)  3 .16reg .D) ;
 6   : S)  4 .16reg .D) ;
 7   : U)  5 .16reg .D) ;
 8
 9   : U+W)  5 .16reg  7 .16reg .I) ;
10   : R+I)  3 .16reg  6 .16reg .I) ;
11   : U+I)  5 .16reg  6 .16reg .I) ;
12   : R+W)  3 .16reg  7 .16reg .I) ;
13
14   : .#    ." # " ;
15
```

30

```
 \
  external
  .SELF LES .SELF LDS  .SELF INTO  .SELF IRET
  internal

  : LES/LDS    \ op ---  ¦ les/lds instruction  C4-C5
   NEXTB .MREG  .REG  DROP 1 exec LES LDS STOP[
  external
  : RET        \ op ---  ¦ return instruction  C2-C3, CA-CB
   1 mask 0= IF WDISP ."  SP+" THEN
   8 [and] IF ." FAR " THEN  .myself ;

  internal
  : MOV#R/M    \ op ---  ¦ return instruction  C2-C3, CA-CB
   NEXTB  ?DISP  OVER 1 [and]  IF  WDISP  ELSE  BIMM  THEN .#
   .MREG  MOV 2DROP ;
```

31

```
 \
  external

  : INT        \ op ---  ¦ int instruction  CC-CD
    1 [and] IF NEXTB ELSE 3 THEN U.  .myself ;

  internal
  : INTO/IRET  \ op ---  ¦ int & iret instructions  CE-CF
    1 exec  INTO IRET STOP[

  : COS        \ op ---  ¦ display instructions  C0-CF
    DUP 2/ 7 exec
    OOPS1 RET LES/LDS  MOV#R/M  OOPS1 RET INT  INTO/IRET  STOP[
```

32

```
 \
  external
  .SELF ROL      .SELF ROR      .SELF RCL      .SELF RCR
  .SELF SHL/SAL  .SELF SHR      .SELF SAR
  internal

  : SHIFTS     \ op ---  ¦ secondary instructions d0-d3
   2 mask IF 0 .8reg ( C-) THEN
   NEXTB  .MREG NIP  3 shift> 7 exec
   ROL  ROR  RCL  RCR  SHL/SAL SHR  OOPS0 SAR  STOP[

  : XLAT   DROP ." XLAT " ;

  : ESC        \ op ---  ¦ esc instructions d8-DF
   NEXTB .MREG  3 shift> 7 [and] U.  7 [and] U.  ." ESC " ;
```

1 12

```
 0 \
 1  internal
 2
 3  : (.R/M)     \ op ext ---  ¦ print a register
 4    IM OFF SWAP 1 [and]  IF .16REG exit then .8REG ;
 5
 6  : .R/M       \ op ext --- op ext ¦ print r/m as register
 7    2DUP  (.R/M) ;
 8
 9  : .REG       \ op ext --- op ext ¦ print reg as register
10    2DUP  3 shift> (.R/M) ;
11
12
13
14
15
```

1 13

```
 0 \
 1  internal
 2
 3  CREATE SEGTB ," ECSD"
 4
 5  : (.seg ( n -- )
 6   3 shift> 3 and segtb + 1+ c@ emit ;
 7
 8  : .SEG       \ s# ---  ¦ register printed out
 9   (.seg ." : " ;
10
11  : SEG:       \ op ---  ¦ print segment overrides
12   (.seg ." S:" ;
13
14
15
```

1 14

```
 0 \
 1  internal
 2  : disp@  ( ops-cnt -- )
 3    ops +! CP@  IM @ + IM off ." $"  ;
 4
 5  : BDISP   \  ---  ¦ do if displacement is byte
 6   1 disp@ TC@ sext U. ;
 7
 8  : WDisp   \  ---  ¦ do if displacement is word
 9   2 disp@ T@  U. ;
10
11  : .DISP      \ op ext --- op ext ¦ print displacement
12   DUP 6 shift>  3 EXEC  noop BDISP WDISP .R/M STOP[
13
14  : BIMM       \  ---  ¦ do if immed. value is byte
15   1 disp@  TC@ . ;
```

33

```
 \
 internal
 : DOS        \ op ---  ¦ display instructions  D0-DF
  8 mask IF ESC EXIT THEN
  DUP 7 exec
  SHIFTS SHIFTS SHIFTS SHIFTS  .AAM .AAD OOPS1 XLAT STOP[

 external
 .SELF LOOPE/Z  .SELF LOOP  .SELF JCXZ   .SELF LOOPNE/NZ
 internal

 : LOOPS      \ op ---  ¦ display instructions  E0-E3
  NEXTB SEXT  CP@ + u. 3 exec
  LOOPNE/NZ  LOOPE/Z  LOOP  JCXZ  STOP[

 external .SELF IN  .SELF OUT   .SELF JMP
```

34

```
 \
 internal

 : IN/OUT     \ op ---  ¦ display instructions  E4-E6,EC-EF
  8 mask
   IF    2 mask IF .AL/AX .D  OUT ELSE .D .AL/AX  IN THEN
   ELSE  2 mask
         IF .AL/AX BIMM .# OUT ELSE BIMM .# .AL/AX IN  THEN
   THEN ;
```

35

```
 \
 internal
 : CALLs       \ op ---  ¦ display instructions  E7-EB
  2 mask IF  1 mask IF  NEXTB SEXT  CP@ + u.
                    ELSE  INTER  THEN
         ELSE  NEXTW CP@ + u. THEN
  3 exec CALL JMP JMP  JMP STOP[

 : EOS        \ op ---  ¦ display instructions  E0-EF
  DUP 2 shift>  3 EXEC  LOOPS  IN/OUT CALLs  IN/OUT STOP[

 : FTEST      \ op ---  ¦ display instructions  F6,7:0
  ?DISP OVER 1 [and] IF WDISP ELSE BIMM THEN .#
  .MREG DROP  .SIZE  0 .ss-tab ; \ TEST
```

1 15

```
 0 \
 1  internal
 2
 3
 4  : .MREG   \ op ext --- op ext ¦ register(s) printed out + disp
 5    $C7 mask 6 = IF  WDISP ." ) " exit then
 6    $C0 mask $C0 - 0= IF .R/M exit THEN
 7    .DISP DUP  7 exec
 8      R+I)  R+W)  U+I)  U+W)      \ I) oder DI)
 9        I)    W)    U)    R)      \  ) oder  D)
10  ;
11
12
13
14
15
```

1 16

```
 0 \
 1  internal
 2
 3  : .SIZE      \ op ---  ¦ decodes for size; WORD is default
 4    1 [and] 0= IF ." BYTE " THEN ;
 5
 6  create adj-tab ," DAADASAAAAASAAMAAD"
 7
 8  : .adj-tab 3 * adj-tab 1+ + 3 type space  ;
 9
10  : ADJUSTS    \ op ---  ¦ the adjusts
11    3 shift>  3 [and] .adj-tab ;
12
13  : .AAM  4 .adj-tab  nextb 2drop ;
14  : .AAD  5 .adj-tab  nextb 2drop ;
15
```

1 17

```
 0 \
 1  internal
 2  : .POP       \ op ---  ¦ print pops
 3    DUP 8 = IF OOPS1 THEN  .SEG  ." POP " ;
 4
 5  : .PUSH      \ op ---  ¦ print pushes
 6    .SEG  ." PUSH " ;
 7
 8  : P/P        \ op --- ¦ pushes or pops
 9   1 mask IF .pop ELSE .push THEN ;
10
11
12
13
14
15
```

36

```
 \
  external
  .SELF NOT   .SELF NEG   .SELF MUL      .SELF IMUL
  .SELF DIV   .SELF IDIV  .SELF REP/NZ   .SELF REPZ
  .SELF LOCK  .SELF HLT   .SELF CMC      .SELF CLC
  .SELF STC   .SELF CLI   .SELF STI      .SELF CLD
  .SELF STD   .SELF INC   .SELF DEC      .SELF PUSH
  internal

 : MUL/DIV    \ op ext ---  ¦ secondary instructions F6,7:4-7
   .MREG  .A OVER 1 [and] IF .D THEN  NIP
   3 shift> 3 exec MUL IMUL DIV IDIV STOP[
```

37

```
 \
  internal
  : NOT/NEG    \ op ext ---  ¦ secondary instructions F6,7:2,3
   .MREG SWAP .SIZE  3 shift> 1 exec NOT NEG STOP[

  : F6-F7S     \ op ---  ¦ display instructions  F6,7
   NEXTB  DUP 3 shift>  7 exec   FTEST OOPS2 NOT/NEG NOT/NEG
   MUL/DIV MUL/DIV MUL/DIV MUL/DIV  STOP[

  : FES        \ op ---  ¦ display instructions  FE
   NEXTB .MREG ." BYTE " NIP 3 shift>
    3 exec INC DEC oops oops STOP[

  : FCALL/JMP  \ op ext ---  ¦ display call instructions  FF
   .MREG  3 shift> 1 mask IF  ." FAR "  THEN
   NIP  2/  1 exec  JMP CALL  STOP[
```

38

```
 \
  internal

  : FPUSH      \ op ext ---  ¦ display push instructions  FF
   dup $FF = IF oops2 exit THEN    \ FF FF gibt's nicht!
   4 mask IF .MREG  2DROP PUSH EXIT THEN  OOPS2 ;

  : FINC       \ op ext ---  ¦ display inc/dec instructions  FF
   .MREG  NIP 3 shift> 1 exec INC DEC STOP[

  : FFS        \ op ---  ¦ display instructions  FF
   NEXTB DUP 4 shift> 3 exec
   FINC  FCALL/JMP FCALL/JMP  FPUSH   STOP[
```

1 18

```
 0 \
 1 internal
 2  : P/SEG      \ op ---  ¦ push or seg overrides
 3    DUP 5 shift>  1 exec  P/P SEG: STOP[
 4
 5  : P/ADJ      \ op ---  ¦ pop or adjusts
 6    DUP 5 shift>  1 exec  P/P ADJUSTS  STOP[
 7
 8  : OGP        \ op --- op ¦ opcode decoded & printed
 9    4 mask IF 1 mask
10              IF WDISP ELSE BIMM THEN  .#
11             1 [and] IF .A ELSE .A- THEN  ELSE
12    NEXTB  OVER 2 [and]
13           IF  .MREG .REG  ELSE  ?DISP .REG .MREG
14    THEN  2DROP THEN  ;
15
```

1 19

```
 0 \
 1  external
 2  .SELF ADD  .SELF ADC  .SELF AND  .SELF XOR
 3  .SELF OR   .SELF SBB  .SELF SUB  .SELF CMP
 4
 5  internal
 6
 7  : OGROUP     \ op ---  ¦ select 0 group to print
 8   DUP OGP 3 shift> 7 EXEC
 9   ADD  OR   ADC  SBB  AND  SUB  XOR  CMP  STOP[
10
11  : LOWS       \ op ---  ¦  0-3f opcodes printed out
12   DUP  7 EXEC
13   OGROUP  OGROUP  OGROUP  OGROUP
14   OGROUP  OGROUP  P/SEG   P/ADJ  STOP[
15
```

1 20

```
 0 \
 1  internal
 2
 3  : .REGGP     \ op ---  ¦ register group defining word
 4    CREATE  LAST @ ,    DOES>  @ SWAP  .16REG  .name ;
 5
 6  external
 7
 8  .REGGP INC  .REGGP DEC  .REGGP PUSH  .REGGP POP
 9
10  : POPs       \ op ---  ¦ handle illegal opcode for cs pop
11   $38 mask 8 = IF ." illegal" DROP ELSE POP THEN ;
12
13 : REGS       \ op ---  ¦ 40-5f opcodes printed out
14   DUP 3 shift> 3 exec  INC DEC PUSH POPs   STOP[
15
```

39

```
 \
  internal

  : FOS        \ op ---  ¦ display instructions  F0-FF
   &15 mask  7 mask  6 < IF NIP  THEN  -1 exec
    LOCK  OOPSO REP/NZ REPZ  HLT CMC F6-F7S  F6-F7S
    CLC STC CLI STI CLD STD  FES  FFS  STOP[

  : HIGHS   \ op -- ¦ op codes C0 - FF displayed
   DUP 4 shift>  3 exec COS DOS EOS FOS STOP[

  : (INST)  \ op ---  ¦ highest level vector table
   &255 [and]  DUP 6 shift>
   -1 exec LOWS  MEDS  HMEDS  HIGHS  STOP[
```

40

```
 \
  internal

  : INST  \  ---  ¦ display opcode at ip, advancing as needed
   [ disam ] .seg:off
   NEXTB (INST)  OPS @ CP +!  OPS OFF  IM OFF ;

  : (DUMP) \  addr ct ---  ¦ dump as pointed to by reloc
   [ forth ] BOUNDS  ?do I TC@ u. LOOP ;
```

41

```
 \
  internal

  : steps?
   1+ dup &10 mod 0= IF key #esc = exit THEN  0 ;

  create next-code   assembler  next forth

  : ?next ( steps-count -- steps-count )
   cp@ 2@  next-code 2@ D=
   IF  cr .seg:off ." NEXT Link= "  cp@ 4+ @ U.
       cp@ 6 +  cp ! \ 4 bytes code, 2 byte link
       drop 9        \ forces stop at steps?
   THEN ;
```

```
1                    0

 0 \                                                ks 08 aug 88
 1  INCLUDE DISKS.CFG
 2  allows to specify disk capacities interactively.
 3
 4  3 LOADFROM DISKS.CFG
 5  sets the hard-disk capacity according to the information
 6  specified via the FDISK.COM command.
 7
 8  After the capacities have been set, the values can be
 9  made permanent with the SAVESYSTEM command.
10
11
12
13
14
15
```

```
1                    1

 0 \ getting an input number                        ks 08 aug 88
 1  Onlyforth Dos also
 2
 3  : input#  ( <string> -- n )  pad c/l expect
 4     pad span @ 2dup upper pad place
 5     pad nullstring? IF  0 exit  THEN  number drop ;
 6
 7  2 load
 8
 9 cr cr
10 .( Die Konfiguration kann mit SAVESYSTEM <name> abgespeichert we
11 rden.)
12
13
14
15
```

```
1                    0

 0 \                                                ks 08 aug 88
 1  INCLUDE DISKS.CFG
 2  allows to specify disk capacities interactively.
 3
 4  3 LOADFROM DISKS.CFG
 5  sets the hard-disk capacity according to the information
 6  specified via the FDISK.COM command.
 7
 8  After the capacities have been set, the values can be
 9  made permanent with the SAVESYSTEM command.
10
11
12
13
14
15
```

```
2

\ configuring disk capacities                      ks 08 aug 88
  page

  .( volksFORTH unterstützt im DIRECT Modus maximal 6 logische
aufwerke)
cr .( Im folgenden die maximalen Kapazitäten der Laufwerke eing
ben:) cr

  cr .( A: ) input# capacities !
  cr .( B: ) input# capacities 2+ !
  cr .( C: ) input# capacities 4 + !
  cr .( D: ) input# capacities 6 + !
  cr .( E: ) input# capacities 8 + !
  cr .( F: ) input# capacities &10 + !  toss empty
```

```
3

\ Winchester boot sector capacity determination   ks 08 aug 88
  Onlyforth \needs Assembler   2 loadfrom asm.scr

  Code get-boot  ( addr -- f )   $201 # A mov   1 # C mov
     R W mov   D R mov   $80 # D mov   $13 int
     W R mov   0 # D mov  CS not ?[  D dec  ]?  Next
  end-code

  : set-capacities  [ Dos ] 5  BEGIN  pad get-boot
     IF  capacities 4 +   pad [ $1BE $C + ] Literal +
         $40 bounds DO  I @ ?dup IF  2/ over !  THEN  2+
                  $10 +LOOP   2drop exit
     THEN  1- ?dup 0= UNTIL
     true Abort" Bootsector can't be read" ;

  set-capacities   empty
```

```
0

\                                                  ks 08 aug 88
  INCLUDE DISKS.CFG
  allows to specify disk capacities interactively.

  3 LOADFROM DISKS.CFG
  sets the hard-disk capacity according to the information
  specified via the FDISK.COM command.

  After the capacities have been set, the values can be
  made permanent with the SAVESYSTEM command.
```

1 0

```
 0 \                                                    28 jun 88
 1
 2 DOS loads higher level file functions which go beyond
 3 including a screen file. Calls to MS-DOS are implemented
 4 and used for directory manipulation. These functions may
 5 not work for versions before MS-DOS 3.0.
 6
 7
 8
 9
10
11
12
13
14
15
```

1 1

```
 0 \ MS-DOS file handli                                 28 jun 88
 1  Onlyforth  \needs Assembler  2 loadfrom asm.scr
 2
 3  : fswap     isfile@  fromfile @  isfile !  fromfile ! ;
 4
 5  $80 Constant dta
 6
 7 ¦ : COMSPEC  ( -- string )  [ dos ]
 8     $2C @ ( DOS-environment:seg) 8  ds@ filename  &60 lmove
 9     filename counted &60 min filename place  filename ;
10
11  1 &12 +thru  .( Dos Funktionen geladen) cr
12
13  Onlyforth
14
15
```

1 2

```
 0 \ moving blocks                                      ks 04 okt 87
 1
 2 ¦ : full? ( -- flag )  prev  BEGIN @ dup @ 0= UNTIL  6 + @ 0< ;
 3
 4  : used?  ( blk -- f )
 5     block count b/blk 1- swap skip nip 0<> ;
 6
 7 ¦ : (copy  ( from to -- )
 8     full? IF  save-buffers  THEN  isfile@ fromfile @ -
 9     IF  dup used? Abort" target block not empty"  THEN
10     dup isfile@ core? IF  prev @ emptybuf  THEN
11     isfile@ 0= IF  offset @ +  THEN
12     isfile@ rot  fromfile @ (block 6 -  2! update ;
13
14
15
```

8

```
\ getpath                                              ks 10 okt 87
  Dos definitions

¦ &40 Constant pathlen
¦ Create pathes  0 c, pathlen allot

¦ : (setpath  ( string -- )  count
    dup pathlen u> Abort" path too long"  pathes place ;

¦ : getpath  ( +n -- string / ff )
    >r  0  pathes count  r> 0
    DO  rot drop Ascii ; skip  stash  Ascii ; scan  LOOP
    drop over - ?dup
    IF  here place  here  dup count + 1- c@
       ?" :\" ?exit  Ascii \ here append exit
    THEN  0= ;
```

9

```
\ pathsearch  .path  path                              ks 09 okt 87

  : pathsearch  ( string -- asciz *f )  dup >r
    (fsearch dup 0= IF  rdrop exit  THEN  2drop  0 0
    BEGIN  drop 1+  dup getpath ?dup 0=
          IF  drop r> filename >asciz 2 exit  THEN
          r@ count 2 pick attach (fsearch
    0= UNTIL nip rdrop false ;

  ' pathsearch Is fsearch

  Forth  definitions

  : .path    pathes count type ;

  : path     name nullstring? IF  .path exit  THEN  (setpath ;
```

10

```
\ call another executable file
  Dos definitions                                      ks 04 aug 87

¦ Create cpb  0 , \ inherit parent environment
   dta , ds@ , $5C , ds@ , $6C , ds@ ,  Label ssave  0 ,

¦ Code ~exec  ( asciz -- *f )
    I push  R push  U push  S ssave #) mov  cpb # R mov
    $4B00 # A mov  $21 int  C: D mov  D D: mov  D S: mov
    D E: mov  ssave #) S mov  CS not
    ?[  A A xor  A push  $2F # A+ mov  $21 int  E: A mov
       A D: mov  C: A mov  A E: mov  R I mov  dta # W mov
       $40 # C mov  rep movs  A D: mov  A pop
    ]?  A W xchg  dta # D mov  $1A # A+ mov  $21 int
    W D mov  U pop  R pop  I pop  Next
end-code
```

1 3

```
0 \ moving blocks                                   ks 04 okt 87
1
2 : blkmove ( from to quan -- ) 3 arguments save-buffers
3    >r over r@ + over u> >r 2dup u< r> and
4    IF    r@ r@ d+ r> 0 ?DO  -1 -2 d+   2dup (copy  LOOP
5    ELSE          r> 0 ?DO  2dup (copy     1 1 d+  LOOP
6    THEN  save-buffers 2drop ;
7
8  : copy    ( from to -- )        1 blkmove ;
9
10 : convey  ( blk1 blk2 to.blk -- )
11    3 arguments   >r   2dup swap - >r
12    fswap dup capacity 1- > isfile@ 0<> and
13    fswap r> r@ + capacity 1- > isfile@ 0<> and or >r
14    1+ over - dup 0> not r> or Abort" nein" r> swap blkmove ;
15
```

1 4

```
0 \ MORE  extending forth files                     ks 10 okt 87
1  Dos also definitions
2
3 : addblock   ( blk -- )   dup buffer  dup b/blk blank
4    isfile@ f.size dup  2@ b/blk 0 d+  rot 2!
5    swap isfile@ fblock! ;
6
7  Forth definitions
8
9  : more   ( n -- )  1 arguments  isfile@
10    IF  capacity swap bounds ?DO  I addblock  LOOP  close exit
11    THEN  drop ;
12
13
14
15
```

1 5

```
0 \ file  eof? create   dta-addressing              ks 03 apr 88
1  Dos definitions
2
3  : ftime  ( -- mm hh )
4    isfile@ f.time @ $20 u/mod nip $40 u/mod ;
5
6  : fdate  ( -- dd mm yy )
7    isfile@ f.date @ $20 u/mod $10 u/mod &80 + ;
8
9  : .when  base push decimal
10    fdate rot 3 .r ." ." swap 2 .r ." ." 2 .r
11    ftime 3 .r ." :" 2 .r ;
12
13
14
15
```

1 11

```
\ calling MS-DOS thru forth interpreter             ks 19 mär 88

: execute?  ( extension -- *f )
   count   filename count Ascii . scan drop swap
   2dup 1+ erase  move   filename 1+ ~exec ;

: fcall   ( string -- )  count filename place   ds@ cpb 4+ !
   " .EXE" execute? dup IF  drop " .COM" execute?  THEN
   ?diskerror ;

: fdos  ( string -- )
   dta $80 erase   " /c " count dta place   count dta attach
   status push  status off .status COMSPEC fcall  curat? at ;
```

1 12

```
\ einige MS-DOS Funktionen  msdos  call             ks 10 okt 87

: dos:  Create ,"  Does> count here place
   Ascii " parse here attach  here fdos ;

Forth definitions

dos: dir dir "
dos: ren ren "
dos: md md "
dos: cd cd "
dos: rd rd "
dos: fcopy copy "
dos: delete del "
dos: ftype type "
```

1 13

```
\ msdos  call                                       ks 23 okt 88

: msdos  savevideo   status push  status off .status
   flush dta off COMSPEC fcall restorevideo ;

: call    name   source >in @ /string c/l umin
   dta place   dta dta >asciz drop   [compile] \
   status push  status off .status fcall   curat? at ;
```

```
1                6

 0 \                                          ks 20 mär 88
 1
 2  : (.fcb     ( fcb -- )
 3     dup .file ?dup 0=exit  pushfile
 4     isfile !  &13 tab ." is"
 5     isfile@ f.handle @ 2 .r
 6     isfile@ f.size 2@ 7 d.r  .when
 7     space isfile@ f.name count type ;
 8
 9  Forth definitions
10
11  : files   file-link
12     BEGIN  @ dup WHILE  cr dup (.fcb  stop? UNTIL  drop ;
13
14  : ?file    isfile@ (.fcb ;
15
```

```
1                7

 0 \ dir make makefile                        ks 25 okt 87
 1  Forth definitions
 2
 3  : killfile   close
 4     isfile@ f.name filename >asciz ~unlink drop ;
 5
 6  : emptyfile      isfile@ 0=exit
 7     isfile@ f.name filename >asciz  0 ~creat ?diskerror
 8     isfile@ f.handle !    isfile@ f.size 4 erase ;
 9
10  : make      close   name isfile@ fname!   emptyfile ;
11
12  : makefile  File   last @ name> execute   emptyfile ;
13
14
15
```

```
1                0

 0 \                                          28 jun 88
 1
 2 DOS loads higher level file functions which go beyond
 3 including a screen file. Calls to MS-DOS are implemented
 4 and used for directory manipulation. These functions may
 5 not work for versions before MS-DOS 3.0.
 6
 7
 8
 9
10
11
12
13
14
15
```

```
14

 \ time date                                  ks 19 mär 88
   Dos definitions

   : ftime  ( -- mm hh )
      open isfile@ f.time @ $20 u/mod nip $40 u/mod ;

   : fdate  ( -- dd mm yy )
      open isfile@ f.date @ $20 u/mod $10 u/mod &80 + ;
```

```
15

 \ ~lseek  position?                          ks 10 okt 87
   Dos definitions

   Code ~lseek    ( d handle method -- d' )
      R W mov   D A mov   R pop   C pop   D pop
      $42 # A+ mov   $21 int   W R mov   CS not
      ?[  A push   Next  ]?  A D xchg  ;c: ?diskerror ;

   Forth definitions

   : position?   ( -- dfaddr )
      isfile@ f.handle @ 0= Abort" file not open"
      0 0  isfile@ f.handle @  1  ~lseek ;
```

```
0

 \                                            28 jun 88

DOS loads higher level file functions which go beyond
including a screen file. Calls to MS-DOS are implemented
and used for directory manipulation. These functions may
not work for versions before MS-DOS 3.0.
```

```
1                    O

 0 \\ Double words                                  ks 22 dez 87
 1
 2 Dieses File enthaelt Worte fuer 32-Bit Objekte.
 3
 4 Im Kern bereits enthalten sind:
 5
 6    2! 2@ 2drop 2dup 2over 2swap d+ d. d.r
 7    d0= d< d= dabs dnegate
 8
 9
10
11
12
13
14
15
```

```
1

\ 2constant  2rot 2variable d- d2/                  ks 22 dez 87

  : 2constant   Create , ,  does> 2@ ;

  : 2rot   ( d1 d2 d3 -- d2 d3 d1 )  5 roll  5 roll ;

  : 2variable   Variable  2 allot ;

  : d-  ( d1 d2 -- d3 ) dnegate d+ ;

  Code d2/  ( d1 -- d2 )
     A pop   D sar   A rcr   A push   Next   end-code
```

```
1                    O

 0 \\ Double words                                  ks 22 dez 87
 1
 2 Dieses File enthaelt Worte fuer 32-Bit Objekte.
 3
 4 Im Kern bereits enthalten sind:
 5
 6    2! 2@ 2drop 2dup 2over 2swap d+ d. d.r
 7    d0= d< d= dabs dnegate
 8
 9
10
11
12
13
14
15
```

```
O

\\ Double words                                     ks 22 dez 87

Dieses File enthaelt Worte fuer 32-Bit Objekte.

Im Kern bereits enthalten sind:

   2! 2@ 2drop 2dup 2over 2swap d+ d. d.r
   d0= d< d= dabs dnegate
```

```
1                    O

 0 \\ Double words                                  ks 22 dez 87
 1
 2 Dieses File enthaelt Worte fuer 32-Bit Objekte.
 3
 4 Im Kern bereits enthalten sind:
 5
 6    2! 2@ 2drop 2dup 2over 2swap d+ d. d.r
 7    d0= d< d= dabs dnegate
 8
 9
10
11
12
13
14
15
```

```
O

\\ Double words                                     ks 22 dez 87

Dieses File enthaelt Worte fuer 32-Bit Objekte.

Im Kern bereits enthalten sind:

   2! 2@ 2drop 2dup 2over 2swap d+ d. d.r
   d0= d< d= dabs dnegate
```

1 0

```
 0          volksFORTH Full-Screen-Editor HELP Screen
 1
 2 Editor verlassen    : flushed: ESC    updated: ^E
 3 Änderungen verwerfen: ^U  (UNDO)
 4 Cursor bewegen      : Cursortasten   (löschen mit DEL oder <- )
 5 Einfügen            : INS (an/aus),  ^ENTER (Screen einfügen)
 6 Tabs                : TAB (nach rechts), SHIFT TAB (nach links)
 7 Blättern            : Pg Dn (nächster),  Pg Up (voriger)
 8                     : F9    (alternate), SHIFT F9 (shadow)
 9 mark alternate Scr. : F10
10 Zeile löschen/einf. : ^Y (löschen), ^N (einfügen)
11 Zeile teilen        : ^PgDn (split), ^PgUp (join)
12 Suchen und Ersetzen : F2 (Break mit ESC, replace mit 'R' )
13 Zeilenpuffer        : F3 (push&delete), F5 (push), F7 (pop)
14 Zeichenpuffer       : F4 (push&delete), F6 (push), F8 (pop)
15 Sonstige            : ^F (Fix), ^L (Showload), ^S (Screen #)
```

1 1

```
 0 -->    \ Full-Screen Editor                ks 22 dez 87
 1 Dieses File enthaelt den Full-Screen Editor fuer die IBM -
 2 volksFORTH-Version.
 3
 4 Er enthaelt Line- und Chararcter-Stacks, Find&Replace-Funktion
 5 sowie Unterstuetzung des Shadow-Screen-Konzepts, der view-
 6 Funktion und des sichtbaren Laden von Screens (showload).
 7
 8 Durch die integrierte Tastaturtabelle (keytable) laesst sich die
 9 Kommandobelegung der Tasten auf einfache Art und Weise aendern.
10
11 Angepaßt für den IBM PC von K.Schleisiek am 22 dez 87
12 Anregungen, Kritik und Verbesserungsvorschlaege bitte an:
13                      U. Hoffmann
14                      Harmsstrasse 71
15                      2300 Kiel
```

1 2

```
 0 \ Load Screen for the Editor                      UH 11dez88
 1
 2   Onlyforth     \needs Assembler   2 loadfrom asm.scr
 3
 4       3 load    \ PC adaption
 5   4   9 thru    \ Editor
 6
 7 \    &10 load   \ ANSI display interface
 8 \    &11 load   \ BIOS display interface
 9      &12 load   \ MULTItasking display interface
10
11 &13 &39 thru    \ Editor
12
13 Onlyforth  .( Screen Editor geladen) cr
14
15
```

20

```
 \ join and split lines                                  UH 11dez8
 : insert-spaces ( n -- )  'cursor swap
    2dup over #remaining insert  blank ;
 : split ( -- )  ?bottom cursor  col#  <cr> insert-spaces  r#
    #after insert-spaces  screenmodified ;
 : delete-characters ( n -- ) 'cursor #remaining rot delete ;
 : join ( -- ) cursor <cr>  line> col#  <line col# under -
    rot r# ! #after > Abort" next line will not fit!"
    #after +  dup delete-characters
    cursor  <cr> c/l rot - dup 0<
    IF negate insert-spaces ELSE delete-characters THEN r# !
    screenmodified ;
```

21

```
 \ handle characters                                     UH 01Nov8
 : delete-char   'cursor #after 1 delete  linemodified ;
 : backspace   curleft  delete-char ;
 : (insert-char   ?end  'cursor 1 over #after insert ;
 : insert-char   (insert-char  bl 'cursor c! linemodified ;
 : putchar  ( --)  char c@
    imode @ IF (insert-char THEN
    'cursor c!  linemodified  curright ;
```

22

```
 \ stack lines                                           UH 31Oct8
 Create lines 4 allot    \  { 2+pointer ¦ 2base }
 : 'lines  ( -- adr)   lines 2@ + ;
 : @line   'lines  memtop u> Abort" line buffer full"
    'line 'lines c/l cmove  c/l lines +! ;
 : copyline   @line curdown ;
 : line>buf   @line delete-line ;
 : !line   c/l negate lines +!  'lines 'line c/l cmove ;
 : buf>line   lines @ 0= Abort" line buffer empty"
    ?bottom    (insert-line  !line   screenmodified ;
```

1 3

```
 0 \ BIM adaption                                        UH 11dez88
 1
 2 ¦ : ?range ( n -- n ) isfile@ 0=exit dup 0< 9 and ?diskerror
 3       dup capacity - 1+  0 max  ?dup 0=exit  more  ;
 4 ¦ : block    ( n -- adr )  ?range  block ;
 5
 6   $1B Constant #esc
 7
 8   : curon    &11 &12 curshape ;
 9
10   : curoff   &14 dup curshape ;
11
12   Variable caps   caps off
13
14   Label ?capital   1 # caps #) byte test
15      0= ?[  (capital # jmp  ]?   ret   end-code
```

1 4

```
 0 \ search delete insert replace                    ks 20 dez 87
 1
 2 ¦ : delete   ( buffer size count -- )
 3      over min >r  r@ - ( left over )  dup 0>
 4      IF  2dup swap dup  r@ + -rot swap cmove THEN
 5      + r> bl fill ;
 6
 7 ¦ : insert   ( string length buffer size -- )
 8      rot over min >r  r@ - ( left over )
 9      over dup r@ +  rot cmove>   r> cmove  ;
10
11 ¦ : replace   ( string length buffer size -- )
12      rot min cmove ;
13
14
15
```

1 5

```
 0 \ usefull definitions and Editor vocabulary       UH 11mai88
 1
 2 Vocabulary Editor
 3
 4 ' Forth  ¦ Alias [F] immediate
 5 ' Editor ¦ Alias [E] immediate
 6
 7 Editor also definitions
 8
 9 ¦ : c  ( n --)   \ moves cyclic thru the screen
10      r# @ +  b/blk mod  r# ! ;
11
12 ¦ Variable r#'      r#'     off
13 ¦ Variable scr'     scr'    off
14 ' fromfile ¦ Alias isfile'
15 ¦ Variable lastfile  ¦ Variable lastscr   ¦ Variable lastr#
```

1 23

```
 0 \ stack characters                                    UH 01Nov8
 1
 2 ¦ Create chars 4 allot    \  { 2+pointer ¦ 2base }
 3 ¦ : 'chars  ( -- adr)   chars 2@ + ;
 4
 5 ¦ : @char   'chars 1-  lines 2+ @  u> Abort" char buffer full"
 6      'cursor c@ 'chars c!  1 chars +! ;
 7
 8 ¦ : copychar   @char curright ;
 9 ¦ : char>buf   @char delete-char ;
10
11 ¦ : !char   -1 chars +!  'chars c@  'cursor c! ;
12
13 ¦ : buf>char   chars @ 0= Abort" char buffer empty"
14      ?end  (insert-char  !char  linemodified ;
```

1 24

```
 0 \ switch screens                                      UH 11mai8
 1
 2 ¦ : imprint ( -- ) \ remember valid file
 3       isfile@ lastfile !  scr @ lastscr !   r# @ lastr# ! ;
 4
 5 ¦ : remember ( -- )
 6       lastfile @ isfile !  lastscr @ scr !  lastr# @ r# ! ;
 7
 8 ¦ : associate \ switch to alternate screen
 9      isfile' @ isfile@   isfile' ! isfile !
10      scr' @ scr @ scr' ! scr !   r#' @ r# @ r#' ! r# ! ;
11
12 ¦ : mark   isfile@ isfile' !  scr @ scr' !  r# @ r#' ! .all ;
13 ¦ : n   ?stamp   1 scr +! .all ;
14 ¦ : b   ?stamp  -1 scr +! .all ;
15 ¦ : a   ?stamp associate  .all ;
```

1 25

```
 0 \ shadow screens                                      UH 03Nov8
 1
 2   Variable shadow   shadow off
 3
 4 ¦ : (shadow   isfile@ IF capacity 2/ exit THEN  shadow @ ;
 5
 6 ¦ : >shadow  ?stamp   \ switch to shadow screen
 7       (shadow dup  scr @  u> not IF negate THEN scr +! .all ;
```

```
1                6

 0 \\ move cursor with position-checking          ks 18 dez 87
 1 \ different versions of cursor positioning error reporting
 2
 3 : c  ( n --)   \ checks the cursor position
 4      r# @ +  dup 0 b/blk uwithin not
 5       Abort" There is a border!"  r# ! ;
 6
 7 : c  ( n --)   \ goes thru the screens
 8    r# @ +  dup b/blk 1- >  IF  1 scr +!  THEN
 9    dup 0< IF  -1 scr +!  THEN  b/blk mod  r# ! ;
10
11 : c  ( n --)   \ moves cyclic thru the screen
12    r# @ +  b/blk mod  r# ! ;
13
14
15
```

```
1                7

 0 \ calculate addresses                          ks 20 dez 87
 1 : *line      ( l -- adr )  c/l * ;
 2 : /line      ( n -- c l )  c/l /mod ;
 3 : top        ( --       )  r# off ;
 4 : cursor     ( -- n     )  r# @ ;
 5 : 'start     ( -- adr )    scr @ block ;
 6 : 'end       ( -- adr )    'start  b/blk + ;
 7 : 'cursor    ( -- adr )    'start  cursor  + ;
 8 : position   ( -- c l )    cursor  /line ;
 9 : line#      ( -- l     )  position nip ;
10 : col#       ( -- c     )  position drop ;
11 : 'line      ( -- adr )    'start  line#  *line + ;
12 : 'line-end  ( -- adr )    'line c/l + 1- ;
13 : #after     ( -- n     )  c/l col# - ;
14 : #remaining ( -- n     )  b/blk cursor - ;
15 : #end       ( -- n     )  b/blk line# *line - ;
```

```
1                8

 0 \ move cursor directed                    UH 11dez88
 1 Create >at 0 , 0 ,
 2 : curup      c/l negate c ;
 3 : curdown    c/l c ;
 4 : curleft    -1 c ;
 5 : curright   1 c ;
 6
 7 : +tab  ( 1/4 -> ) cursor  $10 / 1+ $10 * cursor - c ;
 8 : -tab ( 1/8 <- ) cursor  8 mod  negate  dup 0=  8 * +  c ;
 9
10 : >last  ( adr len -- ) -trailing nip  b/blk min r# ! ;
11 : <cr>   #after c ;
12 : <line ( -- )   col# negate c  'line c/l -trailing nip 0=exit
13      BEGIN 'cursor c@ bl = WHILE curright REPEAT ;
14 : line> ( -- )   'start  line# 1+ *line 1- >last ;
15 : >""end ( -- )  'start  b/blk  >last ;
```

```
1               26

 \ load and show screens                          ks 02 mär 88

 : showoff  ['] exit 'name !  normal  ;

 : show ( -- )  blk @ 0= IF  showoff exit  THEN
    >in @ 1- r# !  edit-at  imprint   blk @  scr @ - 0=exit
    blk @  scr ! normal curoff .all invers curon ;

 : showload  ( -- )  ?stamp  save-buffers
    ['] show 'name !  curon invers
    adr .status push  ['] noop is .status
    scr @  scr push  scr off  r# push  r# @  (load  showoff ;
```

```
1               27

 \ find strings                                   ks 20 dez 87
 Variable insert-buffer
 Variable find-buffer

 : 'insert ( -- addr )  insert-buffer @ ;
 : 'find   ( -- addr )  find-buffer @ ;

 : .buf ( addr -- )   count type ." |"  &80 col - spaces  ;

 : get ( addr -- )  >r at?  r@ .buf
    2dup at  r@ 1+ c/l expect  span @ ?dup IF  r@ c!  THEN
    at r> .buf ;

 : get-buffers    dy 1/s + 2+  dx 1-  2dup  at
       ." find:    |"  'find  get     swap 1+ swap 2- at
      ." ? replace: |"  'insert get ;
```

```
1               28

 \                                                ks 20 dez 87
 Code match    ( addr1 len1 string -- addr2 len2 )
    D W mov   W ) D- mov   $FF # D and  0= ?[  D pop   Next  ]
    W inc   D dec   C pop   I A mov   I pop   A push
    W ) A- mov   W inc   ?capital # call   A- A+ mov   D C sub
    >= ?[  I inc   Label done   I dec
             A pop   I push   A I mov   C D add   Next  ]?
    [[  byte lods   ?capital # call   A+ A- cmp  0=
       ?[  D D or  done 0= not ?]
          I push   W push   C push   A push   D C mov
          [[  byte lods   ?capital # call   A+ A- xchg
             W ) A- mov   W inc   ?capital # call   A+ A- ca
             0= ?[[  C0= ?]   A pop   C pop
                              W pop   I pop   done ]]
                ]?  A pop   C pop   W pop   I pop
    ]?  C0= ?]  I inc   done ]]   end-code
```

1 9

```
 0 \ show border                                    UH 29Sep87
 1
 2 &14 ¦ Constant dx       1 ¦ Constant dy
 3
 4 ¦ : horizontal ( row eck1 eck2 -- row' )
 5     rot dup >r dx 1- at  swap emit
 6     c/l 0 DO Ascii - emit LOOP emit  r> 1+ ;
 7
 8 ¦ : vertical ( row -- row' )
 9     1/s 0 DO dup dx 1-    at Ascii ¦ emit
10             row dx c/l + at Ascii ¦ emit    1+   LOOP ;
11
12 ¦ : border  dy 1- Ascii ┌ Ascii ┐ horizontal
13     vertical     Ascii └ Ascii ┘ horizontal drop ;
14
15 ¦ : edit-at ( -- )   position swap dy dx d+ at ;
```

1 10

```
 0 \  ANSI  display interface                       ks 03 feb 88
 1
 2
 3
 4
 5
 6
 7
 8 ¦ : redisplay ( line# -- )
 9     dup dy +  dx at   *line 'start +  c/l type ;
10
11 ¦ : (done ( -- )  ; immediate
12
13
14 ¦ : install-screen ( -- )  1/s 6 + 0 >at 2! page ;
15
```

1 11

```
 0 \  BIOS-display interface                        ks 03 feb 88
 1 ¦ Code (.line  ( line addr videoseg -- )
 2     A pop   W pop   I push   E: push   D E: mov
 3     $0E # W add   W W add   A I xchg   c/l # C mov
 4     attribut #) A+ mov   [[  byte lods   stos   C0= ?]
 5     E: pop   I pop   D pop   Next   end-code
 6
 7
 8 ¦ : redisplay ( line# -- )
 9     dup 1+ c/row * swap c/l * 'start + video@ (.line ;
10
11 ¦ : (done   ( -- )  ; immediate
12
13
14 ¦ : install-screen ( -- )  1/s 6 + 0 >at 2! page ;
15
```

29

```
 \ search for string                                UH 11mai8
 ¦ : skip ( addr -- addr' )  'find c@ + ;

 ¦ : search  ( buf len string -- offset flag )
     >r stash r@ match  r> c@ <
     IF  drop 0= false exit  THEN  swap - true ;

 ¦ : find?  ( -- r# f )  'cursor #remaining 'find search ;

 ¦ : searchthru ( -- r# scr )
    find? IF  skip cursor +  scr @ exit THEN  drop
    capacity   scr @ 1+
    ?DO  I 2 3 at 6 .r   I block b/blk 'find search
     IF  skip I endloop exit  THEN  stop? Abort" Break!"
    LOOP  true Abort" not found!" ;
```

30

```
 \ replace strings                                  UH 14mai8
 ¦ : replace? ( -- f )  dy 1/s + 3+ dx 3 - at
      key dup #cr = IF line# redisplay true Abort" Break!" THEN
      capital Ascii R = ;

 ¦ : "mark ( -- )   r# push
     'find count dup negate c  edit-at  invers type normal ;

 ¦ : (replace  'insert c@  'find c@ - ?fit
      r# push  'find c@ negate c
      'cursor #after  'find c@ delete
      'insert count   'cursor #after  insert  modified ;

 ¦ : "replace   get-buffers  BEGIN searchthru
      scr @ - ?dup IF ?stamp  scr +! .all THEN  r# !  imprint
      "mark  replace? IF (replace THEN  line# redisplay REPEAT
```

31

```
 \ Display Help-Screen, misc commands               UH 11mai8
 ¦ : helpfile ( -- ) fromfile push  editor.scr ;
 ¦ : .help ( --)
     isfile push   scr push   helpfile   scr off .screen ;
 ¦ : help ( -- )  .help  key drop  .screen ;

 ¦ : screen# ( -- scr )  scr @ ;

 ¦ Defer (fix-word

 ¦ : fix-word ( -- )  isfile@ loadfile !
     scr @ blk !  cursor >in !  (fix-word ;
```

1 12

```
 0 \  MULTI-display interface                    ks    UH 10Sep87
 1 ¦ Code (.line  ( line addr videoseg -- )
 2      C pop   W pop   I push   E: push   D E: mov
 3      $0E # W add   W W add   u' area U D) I mov
 4      u' catt I D) A+ mov   C I mov
 5      c/l # C mov   [[  byte lods   stos   C0= ?]
 6      E: pop   I pop   D pop   Next   end-code
 7
 8 ¦ : redisplay ( line# -- )
 9      dup 1+ c/row * swap c/l * 'start + video@ (.line ;
10
11 ¦ : (done ( -- )  line# 2+ c/col 2- window  ;
12
13 ¦ : cleartop ( -- )   0 l/s 5 + window (page ;
14 ¦ : install-screen ( -- )    row l/s 6 + u<
15    IF l/s 6 + 0 full page ELSE at? cleartop THEN >at 2! ;
```

1 13

```
 0 \ display screen                              UH 11mai88
 1 Forth definitions
 2 : updated?  ( -- f)   'start 2- @  0< ;
 3 Editor definitions
 4 ¦ : .updated ( -- )  9 0 at
 5       updated? IF 4 spaces ELSE ." not " THEN ." updated" ;
 6
 7 ¦ : .screen  l/s 0 DO I redisplay LOOP ;
 8 \ ¦ : .file   ( fcb -- )
 9 \      ?dup IF  body> >name .name exit  THEN  ." direct" ;
10 ¦ : .title [ DOS ]  1 0 at isfile@  .file dx 1- tab
11       2 0 at drv (.drv scr @ 6 .r
12       4 0 at fromfile @  .file  dx 1- tab
13       5 0 at fswap  drv (.drv scr' @ 6 .r  fswap  .updated ;
14
15 ¦ : .all   .title .screen ;
```

1 14

```
 0 \ check errors                                UH 02Nov86
 1
 2 ¦ : ?bottom  ( -- )  'end  c/l -  c/l -trailing nip
 3                      Abort" You would lose a line" ;
 4
 5 ¦ : ?fit ( n -- )  'line c/l -trailing  nip + c/l >
 6      IF line# redisplay
 7         true Abort" You would lose a char" THEN ;
 8
 9 ¦ : ?end    1 ?fit ;
10
11
12
13
14
15
```

32

```
\ Control-Characters  IBM-PC Functionkeys          UH 10Sep
Forth definitions

: Ctrl ( -- c )
   name 1+ c@  $1F and   state @ IF [compile] Literal THEN ;
immediate

\needs #del $7F Constant #del

Editor definitions

¦ : flipimode    imode @ 0= imode ! .imode ;

¦ : F ( # -- 16b )  $FFC6 swap - ;
¦ : shift ( n -- n' )  dup 0< + &24 - ;
```

33

```
\ Control-Characters  IBM-PC Functionkeys          UH 11dez8

Create keytable
-&72   ,      -&75   ,      -&80   ,      -&77   ,
 3 F   ,      4 F    ,      7 F    ,      8 F    ,
Ctrl F ,      Ctrl S ,      5 F    ,      6 F    ,
 1 F   ,      Ctrl H ,        #del ,      -&83   ,
                             Ctrl Y ,      Ctrl N ,
-&82   ,
   #cr ,        #tab ,   #tab shift ,
 -&119 ,      -&117  ,       2 F   ,      Ctrl U ,
Ctrl E ,        #esc ,      Ctrl L ,    9 F shift ,
-&81   ,      -&73   ,      9 F    ,     &10 F    ,
-&71   ,      -&79   ,     -&118   ,     -&132    ,
#lf    ,
here keytable - 2/ Constant #keys
```

34

```
\ Try a screen Editor                              UH 11dez8

Create: actiontable
curup          curleft        curdown        curright
line>buf       char>buf       buf>line       buf>char
fix-word       screen#        copyline       copychar
help           backspace      backspace      delete-char
( insert-char )               delete-line    insert-line
flipimode                    ( clear-line    clear> )
<cr>           +tab           -tab
top            >""end         "replace       undo
update-exit    flushed-exit   showload       >shadow
n              b              a              mark
<line          line>          split          join
new-screen ;
here actiontable -  2/ 1-   #keys -  abort( # of actions)
```

1 15

```
 0 \ programmer's id                                  ks 18 dez 87
 1
 2 $12 ¦ Constant id-len
 3 Create id   id-len allot   id id-len erase
 4
 5 ¦ : stamp ( -- )  id 1+ count  'start c/l + over - swap  cmove ;
 6
 7 ¦ : ?stamp   ( -- )   updated? IF stamp THEN  ;
 8
 9 ¦ : ## ( n -- )  base push decimal 0 <# # # #> id 1+ attach ;
10
11 ¦ : get-id   ( -- )  id c@ ?exit ID on
12    cr ." Enter your ID : "  at?  3 0 DO Ascii . emit LOOP  at
13    id 2+ 3 expect normal  span @ dup id 1+ c!  0=exit
14    bl id 1+ append  date@ rot ## swap >months id 1+ attach ## ;
15
```

1 16

```
 0 \ update screen-display                          UH 28Aug87
 1
 2 ¦ : emptybuf    prev @  2+ dup on  4+ off ;
 3
 4 ¦ : undo  emptybuf  .all ;
 5
 6 ¦ : modified   updated? ?exit  update .updated ;
 7
 8 ¦ : linemodified   modified  line# redisplay ;
 9
10 ¦ : screenmodified    modified
11      1/s line# ?DO I redisplay LOOP ;
12
13 ¦ : .modified ( -- )  >at 2@ at  space  scr @ .
14       updated? not IF  ." un"  THEN  ." modified"  ?stamp ;
15
```

1 17

```
 0 \ leave editor                                   UH 10Sep87
 1 ¦ Variable (pad   (pad off
 2 ¦ : memtop  ( -- adr)   sp@ $100 - ;
 3
 4 ¦ Create char  1 allot
 5 ¦ Variable imode  imode off
 6 ¦ : .imode   at?  7 0 at
 7       imode @ IF ." insert   "  ELSE ." overwrite" THEN at ;
 8 ¦ : setimode   imode on  .imode ;
 9 ¦ : clrimode   imode off .imode ;
10
11 ¦ : done ( -- )  (done
12      ['] (quit is 'quit  ['] (error errorhandler !  quit ;
13
14 ¦ : update-exit  ( -- )  .modified  done ;
15 ¦ : flushed-exit ( -- )  .modified  save-buffers  done ;
```

35

```
 \ find keys                                        ks 20 dez 87

 ¦ : findkey  ( key -- adr/default )
   #keys 0 DO  dup  keytable [F] I 2* + @ =
      IF drop   [E] actiontable  [F] I 2* + @ endloop exit THEN
   LOOP drop ['] putchar ;
```

36

```
 \ allocate buffers                                 UH 01Nov8

   c/l 2* ¦ Constant cstack-size

   ¦ : nextbuf ( adr -- adr' )   cstack-size + ;

   ¦ : ?clearbuffer   pad (pad @ = ?exit
        pad     dup (pad !
        nextbuf dup find-buffer   !   'find off
        nextbuf dup insert-buffer !   'insert off
        nextbuf dup 0 chars 2!
        nextbuf     0 lines 2! ;
```

37

```
 \ enter and exit the editor, editor's loop         UH 11mai8

   ¦ Variable jingle  jingle on  ¦ : bell  07 charout jingle off ;

   ¦ : clear-error ( -- )
       jingle @ ?exit  dy 1/s + 1+ dx  at c/l spaces jingle on ;

   ¦ : fullquit ( -- ) BEGIN ?clearbuffer edit-at  key dup char c!
        findkey imprint execute ( .status )  clear-error REPEAT ;

   ¦ : fullerror  ( string -- )  jingle @ IF bell THEN  count
       dy 1/s + 1+ over 2/ dx $20 + swap - at invers type normal
       &80 col - spaces  remember  .all quit ;

   ¦ : install   ( -- )
       ['] fullquit Is 'quit  ['] fullerror errorhandler !  ;
```

1 18

```
 0 \ handle screens                                  UH 21jan89
 1
 2 ¦ : insert-screen  ( scr -- )  \ before scr
 3     1 more   fromfile push  isfile@ fromfile !
 4     capacity 2- over 1+  convey ;
 5
 6 ¦ : wipe-screen ( -- )  'start b/blk blank ;
 7
 8 ¦ : new-screen ( -- )
 9     scr @  insert-screen wipe-screen  top screenmodified ;
10
11
12
13
14
15
```

1 19

```
 0 \ handle lines                                    UH 01Nov86
 1
 2 ¦ : (clear-line   'line c/l blank ;
 3 ¦ : clear-line   (clear-line linemodified ;
 4
 5 ¦ : clear>   'cursor #after blank  linemodified ;
 6
 7 ¦ : delete-line   'line #end c/l  delete screenmodified ;
 8
 9 ¦ : backline   curup delete-line ;
10
11 ¦ : (insert-line
12      ?bottom  'line c/l over #end insert (clear-line ;
13
14 ¦ : insert-line   (insert-line  screenmodified ;
15
```

1 0

```
 0          volksFORTH Full-Screen-Editor HELP Screen
 1
 2 Editor verlassen    : flushed: ESC    updated: ^E
 3 Änderungen verwerfen: ^U  (UNDO)
 4 Cursor bewegen      : Cursortasten   (löschen mit DEL oder <- )
 5 Einfügen            : INS (an/aus),  ^ENTER (Screen einfügen)
 6 Tabs                : TAB (nach rechts), SHIFT TAB (nach links)
 7 Blättern            : Pg Dn (nächster),  Pg Up (voriger)
 8                     : F9    (alternate), SHIFT F9 (shadow)
 9 mark alternate Scr. : F10
10 Zeile löschen/einf. : ^Y (löschen), ^N (einfügen)
11 Zeile teilen        : ^PgDn (split), ^PgUp (join)
12 Suchen und Ersetzen : F2 (Break mit ESC, replace mit 'R' )
13 Zeilenpuffer        : F3 (push&delete), F5 (push), F7 (pop)
14 Zeichenpuffer       : F4 (push&delete), F6 (push), F8 (pop)
15 Sonstige            : ^F (Fix), ^L (Showload), ^S (Screen #)
```

38

```
\ enter and exit the Editor                          UH 11mai8

Forth definitions

: v   ( -- )
   [E] 'start drop  get-id   install-screen
   install  ?clearbuffer
   border .all .imode .status quit ;

  ' v Alias ed

: l   ( scr -- )   1 arguments scr !  [E] top  [F] v ;

  ' l Alias edit
```

39

```
\ savesystem    enhanced view                        UH 24jun8

: savesystem   [E] id off  (pad off   savesystem ;

Editor definitions
¦ : >find  ?clearbuffer  >in push
     name  dup c@ 2+ >r  bl over c! r>  'find place ;

Forth definitions
: fix    [ Dos ]  >find ' @view >file
   isfile !  scr !   [E] top curdown
   find? IF  skip 1-  THEN  c v ;

' fix Is (fix-word
```

O

volksFORTH Full-Screen-Editor HELP Screen

Editor verlassen : flushed: ESC updated: ^E
Änderungen verwerfen: ^U (UNDO)
Cursor bewegen : Cursortasten (löschen mit DEL oder <-)
Einfügen : INS (an/aus), ^ENTER (Screen einfügen)
Tabs : TAB (nach rechts), SHIFT TAB (nach links)
Blättern : Pg Dn (nächster), Pg Up (voriger)
: F9 (alternate), SHIFT F9 (shadow)
mark alternate Scr. : F10
Zeile löschen/einf. : ^Y (löschen), ^N (einfügen)
Zeile teilen : ^PgDn (split), ^PgUp (join)
Suchen und Ersetzen : F2 (Break mit ESC, replace mit 'R')
Zeilenpuffer : F3 (push&delete), F5 (push), F7 (pop)
Zeichenpuffer : F4 (push&delete), F6 (push), F8 (pop)
Sonstige : ^F (Fix), ^L (Showload), ^S (Screen #)

1 0

```
 0 \\ Printer Interface                                UH 14sep88
 1
 2 Dieses File enthaelt das Printer Interface zwischen volksFORTH
 3 und dem Drucker.
 4
 5 Damit ist es moeglich Source-Texte auf bequeme Art und Weise
 6 in uebersichtlicher Form auszudrucken (6 auf eine Seite).
 7
 8 In Verbindung mit dem Multitasker ist es moeglich, auch Texte im
 9 Hintergrund drucken zu lassen und trotztdem weiterzuarbeiten.
10
11 Diese Druckersteuerung geht auf Ideen von D. Weineck zurück,
12 wurde von U.Hoffmann für das CP/M volksFORTH angepaßt,
13 und von K.Schleisiek verändert.
14
15
```

1 1

```
 0 \ Printer Interface IBM Graphic Printer          ks 3UH 14sep88
 1   Onlyforth
 2   Vocabulary Printer   Printer definitions also
 3
 4   Variable pcol   pcol off
 5   Variable prow   prow off
 6   Variable prints prints off
 7
 8   2 10 thru  .( Interface für EPSON LQ500 geladen. ) cr
 9 \ 11 load    .( Spooler geladen) cr
10
11   : plist  ( scr -- )  prints lock  output push
12      print 10cpi cr list cr 5 lfs  prints unlock ;
13
14   Onlyforth
15
```

1 2

```
 0 \ Printer controls                               ks 2UH 14sep88
 1
 2 ¦ : ctrl:   ( char -- )   Create c,   Does> c@ lst! ;
 3
 4
 5   8   ctrl: ~bs
 6  $D   ctrl: ~cr
 7  $A   ctrl: ~lf
 8  $C   ctrl: ~ff
 9 $1B ¦ ctrl: ESC
10  $F ¦ ctrl: +17cpi
11 $12 ¦ ctrl: -17cpi
12
13
14
15
```

1 6

```
 0 \ Printer output                                    ks 24 mär 88
 1
 2   : +emit   dup (emit pemit ;
 3   : +cr     (cr pcr ;
 4   : +del    (del pdel ;
 5   : +page   (page ppage ;
 6   : +at     2dup (at pat ;
 7
 8 ¦ Output: >printer   pemit pcr tipp pdel ppage pat pat? ;
 9 ¦ Output: +printer  +emit +cr tipp  +del +page +at (at? ;
10
11 Forth definitions
12
13 : print   >printer normal ;
14 : +print  +printer normal ;
15
```

1 7

```
 0 \ Variables and Setup                               ks 09 mai 88
 1
 2   Printer definitions
 3
 4   $00 ¦ Constant logo
 5   ¦ Variable pageno
 6   ¦ Create scr#s   &14 allot  \ enough room for 6 screens
 7
 8 ¦ : header  ( -- )
 9      normal 4 spaces dark  ." Seite " pageno @ 2 .r
10      &13 spaces  ."  volksFORTH83 der FORTH-Gesellschaft eV "
11      5 spaces  file?  -dark 1 pageno +! ~lf ;
12
13
14
15
```

1 8

```
 0 \ Print 2 screens across on a page                  ks 03 apr 88
 1
 2 ¦ : pr  ( scr# -- )   dup  capacity 1- u>
 3      IF  drop logo  THEN  1 scr#s +!  scr#s dup @ 2* + ! ;
 4
 5 ¦ : 2pr  ( scr#1 scr#2 line# -- )
 6      cr  17cpi dup 2 .r space  c/l * >r
 7      pad $101 bl fill  swap block  r@ +  pad c/l cmove
 8      block r> + pad c/l + 1+ c/l cmove
 9      pad $101 -trailing type ;
10
11 ¦ : 2scr  ( scr#1 scr#2 -- )  cr cr normal &17 spaces
12      wide dark over 4 .r  &18 spaces  dup 4 .r  -wide -dark
13      cr  1/s 0 DO  2dup  I 2pr  LOOP  2drop ;
14
15 ¦ : pr-start  ( --)   scr#s off  1 pageno ! ;
```

1 3

```
 0 \ printer controls                                  UH 14sep88
 1
 2 ¦ : #esc:   ( cn..c1 n -- )   Create  dup c, 0 DO c, LOOP
 3   Does> ESC  count bounds DO  I c@ lst!  LOOP ;
 4
 5     $4D 1 ¦ #esc: (12cpi          $67 1 ¦ #esc: (15cpi
 6     $50 1 ¦ #esc: (10cpi
 7
 8   1 $70 2 #esc: prop       0 $70 2 #esc: -prop
 9
10 : 12cpi   -prop (12cpi -17cpi ;
11 : 15cpi   -prop (15cpi -17cpi ;
12 : 10cpi   -prop (10cpi -17cpi ;
13 : 17cpi   -prop (10cpi +17cpi ;
14
15
```

1 4

```
 0 \ printer controls                                  ks 3UH 14sep88
 1
 2     $34 1 #esc: cursive       $35 1 #esc: -cursive
 3   1 $78 2 #esc: nlq         0 $78 2 #esc: standard
 4 ' standard Alias fast       ' standard Alias draft
 5 $31 $57 2 #esc: wide     $30 $57 2 #esc: -wide
 6     $47 1 #esc: dark          $48 1 #esc: -dark
 7     $32 1 #esc: 6/"           $30 1 #esc: 8/"
 8 $31 $2D 2 #esc: +under   $30 $2D 2 #esc: -under
 9
10   : <rand  ( +n -- )  ESC $6C lst! lst! ;
11
12   : lfs   ( +n -- )   0 DO  ^lf  LOOP ;
13
14   : normal    12cpi ^cr ;
15
```

1 5

```
 0 \ Printer output functions                          ks 07 jan 88
 1
 2   : pemit  ( char -- )    1 pcol +!  dup BL u<
 3     IF  $40 or +under lst! -under exit  THEN  lst! ;
 4
 5   : pcr    ^cr ^lf 1 prow +!   pcol off ;
 6
 7   : pdel   ^bs  pcol @ 1- 0 max pcol ! ;
 8
 9   : ppage  ^ff  prow off   pcol off ;
10
11   : pat    ( row col -- )   dup pcol @ -  dup 0< swap
12     abs 0 DO  BL over IF  drop 8  THEN  lst!  LOOP  drop
13     pcol !   prow ! ;
14
15   : pat?   ( -- row col )   prow @   pcol @ ;
```

9

```
 0 \ Printer 6 screens on a page                       ks 03 apr 88
 1
 2 ¦ : pagepr      header  scr#s off  scr#s 2+
 3     3 0 DO  dup @  over 6 + @  2scr  2+  LOOP  drop  page ;
 4
 5 ¦ : shadowpr    header  scr#s off  scr#s 2+
 6     3 0 DO  dup @  over 2+ @  2scr  4 +  LOOP  drop  page ;
 7
 8 ¦ : pr-flush  ( -- f )  \ any screens left over?
 9     scr#s @  dup 0=exit  0<>
10     BEGIN  scr#s @ 5 < WHILE  -1 pr  REPEAT  logo pr ;
11
12 ¦ Variable shadow
13
14 ¦ : full?  ( -- f )  scr#s @ 6 = ;
15
```

10

```
 0 \ Printer 6 screens on a page                       ks 09 mai 88
 1 Forth definitions
 2
 3 : pthru  ( first last -- )  [ Printer ]
 4    prints lock  output push  print  pr-start  1+ swap
 5    ?DO  I pr    full? IF  pagepr  THEN  LOOP
 6    pr-flush IF  pagepr  THEN  prints unlock ;
 7
 8 : document  ( first last -- ) [ Printer ]
 9    isfile@ IF capacity 2/ shadow ! THEN
10    prints lock  output push  print  pr-start  1+ swap
11    ?DO  I pr  I shadow @ + pr   full? IF  shadowpr  THEN  LOOP
12    pr-flush  IF  shadowpr  THEN  prints unlock ;
13
14 : listing    0  capacity 2/ 1- document ;
15
```

11

```
 0 \ Printerspool                                      ks 30 apr 88
 1
 2 \needs Task \\
 3
 4 ¦ Input: noinput    0  false  drop  2drop  ;
 5
 6
 7 noinput  $100 $200 Task spooler  keyboard
 8
 9 : spool   ( from to -- )
10    isfile@  spooler 3 pass  isfile !  pthru  stop ;
11
12
13
14
15
```

1 0

```
 0 \                                                ks 11 mai 88
 1 Dieses File enthält Definitionen, die zum Laden der weiteren
 2 System- und Applikationsfiles benötigt werden.
 3
 4 Unter anderem finden sich hier auch MS-DOS spezifische
 5 Befehle wie zum Beispiel das Allokieren von Speicher-
 6 platz ausserhalb des auf 64k begrenzten Forthsystems
 7 und einige Routinen, die das Arbeiten mit dem Video-
 8 Display erleichtern sowie einige Operatoren zur String-
 9 manipulation.
10
11
12
13
14
15
```

1 1

```
 0 \ loadscreen for often used words                ks 11 mär 89
 1
 2  Onlyforth  \needs Assembler   2 loadfrom asm.scr
 3
 4  ' save-buffers Alias sav
 5
 6  ' name &12 + Constant 'name
 7
 8  ' page Alias cls
 9
10   1 8 +thru  .( Systemerweiterung geladen) cr
11
12
13
14
15
```

1 2

```
 0 \ Postkernel words                               ks 22 dez 87
 1
 2  : blank   ( addr quan -- )   bl fill ;
 3
 4  Code stash ( u1 u2 -- u1 u1 u2 )
 5     S W mov   W ) push   Next   end-code
 6 \ : stash    ( u1 u2 -- u1 u1 u2 )   over swap ;
 7
 8  : >expect  ( addr len -- )  stash expect  span @ over place ;
 9
10  : .field   ( addr len quan -- )
11     over - >r type r> 0 max spaces ;
12
13  : tab  ( n -- )   col - 0 max spaces ;
14
15
```

5

```
 \ postkernel                                       ks 03 aug 87

  c/row c/col * 2* Constant c/dis   \ characters per display

  Code video@    ( -- seg )   D push   R D mov   $F # A+ mov
     $10 int   R D xchg   0 # D- mov   7 # A- cmp
     0= ?[  $B0 # D+ mov  ][  $B8 # D+ add  ]?  Next
  end-code

  : savevideo  ( -- seg / ff )
      [ c/dis b/seg /mod swap 0<> - ] Literal lallocate
      IF  drop false exit  THEN  video@ 0 2 pick 0 c/dis lmove

  : restorevideo  ( seg -- )   ?dup 0=exit
     dup 0 video@ 0 c/dis lmove  lfree drop ;
```

6

```
 \ string operators   append attach                 ks 21 jun 87

 : .stringoverflow   true Abort" String zu lang" ;

  Code append  ( char addr -- )
     D W mov   D pop   W ) A- mov   1 # A- add  CS
     ?[  ;c: .stringoverflow ; Assembler ]?
     A- W ) mov   0 # A+ mov   A W add
     D- W ) mov   D pop   Next   end-code

  Code attach  ( addr len addr1 -- )  D W mov   C pop
     I D mov   I pop   W ) A- mov   A- A+ mov   C- A+ add  CS
     ?[  ;c: .stringoverflow ; Assembler ]?
     A+ W ) mov   A+ A+ xor   A+ C+ mov   A W add   W inc
     rep byte movs   D I mov   D pop   Next   end-code
```

7

```
 \\ string operators   append attach detract        ks 21 jun 87

  : append   ( char addr -- )
     under   count + c!   dup c@ 1+ swap c! ;

  : attach ( addr len addr.to -- )
     >r under   r@ count + swap move   r@ c@ + r> c! ;

  : detract ( addr -- char )
     dup c@ 1- dup 0> and  over c!
     count >r  dup count -rot  swap r> cmove ;
```

1 3

```
0 \ postkernel                                          ks 08 mär 89
1 \ hier sollte END-CODE eigentlich aehem, also z.B. -TRANSIENT
2
3 \needs end-code  : end-code   toss also ;
4
5  : u?     ( addr -- )   @ u. ;
6
7  : adr    ' >body  state @ 0=exit  [compile] Literal ; immediate
8
9  : Abort(  ( f -- )  IF  [compile] .(  true abort"  !"  THEN
10                     [compile] ( ;
11
12 : arguments  ( n -- )
13    depth 1- > Error" zu wenige Parameter" ;
14
15
```

1 4

```
0 \ MS-DOS memory management                            ks 10 okt 87
1
2  Code lallocate  ( pages -- seg ff / rest err# )
3     R push   D R mov   $48 # A+ mov   $21 int  CS
4     ?[  A D xchg   A pop   R push   A R xchg
5     ][  R pop   A push   0 # D mov  ]?   Next   end-code
6
7  Code lfree      ( seg -- err# )
8     E: push   D E: mov   $49 # A+ mov   $21 int  CS
9     ?[  A D xchg  ][  0 # D mov  ]?   E: pop   Next   end-code
10
11
12
13
14
15
```

1 0

```
0 \                                                     ks 11 mai 88
1 Dieses File enthält Definitionen, die zum Laden der weiteren
2 System- und Applikationsfiles benötigt werden.
3
4 Unter anderem finden sich hier auch MS-DOS spezifische
5 Befehle wie zum Beispiel das Allokieren von Speicher-
6 platz ausserhalb des auf 64k begrenzten Forthsystems
7 und einige Routinen, die das Arbeiten mit dem Video-
8 Display erleichtern sowie einige Operatoren zur String-
9 manipulation.
10
11
12
13
14
15
```

8

```
\ ?"  string operator                                  ks 09 feb 88

\ : (?"  ( 8b -- index )  "lit under count rot
\    scan IF  swap - exit  THEN  2drop false ;

¦ Create months  ," janfebmäraprmaijunjulaugsepoktnovdez"

  : >months   ( n -- addr len )   3 * 2- months + 3 ;

¦ Code (?"  ( 8b -- index )
     A D xchg   I ) C- mov   0 # C+ mov   C I add
     I W mov   I inc   std   0<>rep byte scas   cld
     0= ?[  C inc  ]?  C D mov   Next
  end-code

  : ?"   compile (?" ," align ; immediate restrict
```

9

```
\ Conditional compilation                              ks 12 dez 88
¦ Defer cond

  : .THEN ; immediate

  : .ELSE  ( -- )   0
     BEGIN  name nullstring? IF  drop exit  THEN
            find IF  cond -1 case? ?exit  ELSE  drop  THEN
     REPEAT ; immediate

  : .IF ( f -- )  ?exit [compile] .ELSE ; immediate

¦ : (cond  ( n cfa -- n' )
     ['] .THEN case? IF  1- exit  THEN
     ['] .ELSE case? IF  dup 0= + exit  THEN
     ['] .IF   = 0=exit  1+ ; ' (cond is cond
```

O

\ ks 11 mai 88
Dieses File enthält Definitionen, die zum Laden der weiteren
System- und Applikationsfiles benötigt werden.

Unter anderem finden sich hier auch MS-DOS spezifische
Befehle wie zum Beispiel das Allokieren von Speicher-
platz ausserhalb des auf 64k begrenzten Forthsystems
und einige Routinen, die das Arbeiten mit dem Video-
Display erleichtern sowie einige Operatoren zur String-
manipulation.

1 0

```
 0 \ 8086 Assembler                                  ks 19 mär 88
 1
 2 The 8086 Assembler was written by Mike Perry.
 3 To create and assembler language definition, use the defining
 4 word CODE.  It must be terminated with either END-CODE or
 5 its synonym C;.  How the assembler operates is a very
 6 interesting example of the power of CREATE DOES>  Basically
 7 the instructions are categorized and a defining word is
 8 created for each category.  When the nmemonic for the
 9 instruction is interpreted, it compiles itself.
10
11 Fürs volksFORTH lauffähig gemacht von Klaus Schleisiek
12
13 Nicht intensiv getestet, aber
14    CODE TEST  TOS PUSH   1 # TOS MOV   NEXT   END-CODE
15 funktionierte!
```

1 1

```
 0 \ 8086 Assembler                                  ks 19 mär 88
 1 Onlyforth
 2 Vocabulary Assembler
 3 : octal   8 Base ! ;
 4
 5 decimal  1 14 +THRU  clear
 6
 7 Onlyforth
 8
 9   : Code    Create [ Assembler ] here dup 2- !  Assembler ;
10
11 CR .( 8086 Assembler Loaded )
12 Onlyforth
13
14
15
```

1 2

```
 0 \ 8086 Assembler                                  ks 19 mär 88
 1 : LABEL   CREATE ASSEMBLER   ;
 2 \ 232 CONSTANT DOES-OP
 3 \ 3 CONSTANT DOES-SIZE
 4 \ : DOES?   ( IP -- IP' F )
 5 \    DUP DOES-SIZE + SWAP C@ DOES-OP =  ;
 6 ASSEMBLER ALSO DEFINITIONS
 7 : C;   ( -- )   END-CODE   ;
 8 OCTAL
 9 DEFER C,         FORTH ' C,       ASSEMBLER IS C,
10 DEFER ,          FORTH ' ,        ASSEMBLER IS ,
11 DEFER HERE       FORTH ' HERE     ASSEMBLER IS HERE
12 DEFER ?>MARK
13 DEFER ?>RESOLVE
14 DEFER ?<MARK
15 DEFER ?<RESOLVE
```

1 15

```
\ Structured Conditionals                            ks 19 mär 88
HEX
: IF      C,   ?>MARK  ;
: THEN    ?>RESOLVE   ;
: ELSE    0EB IF   2SWAP   THEN   ;
: BEGIN   ?<MARK   ;
: UNTIL   C,   ?<RESOLVE   ;
: AGAIN   0EB UNTIL   ;
: WHILE   IF   ;
: REPEAT   2SWAP   AGAIN   THEN   ;
: DO      # CX MOV   HERE   ;
  : Next  AX lods  AX DI xchg  0 [DI] jmp
     [ Assembler ] here  next-link @ ,   next-link ! ;
\ volksFORTH uses "inline" Next und gelinkte Liste, um alle
\ NEXT, die existieren, für den debugger wiederzufinden.
DECIMAL
```

1 16

1 17

```
\ 8086 Assembler                                          08OCT83HH
LABEL marks the start of a subroutine whose name returns its
  address.
DOES-OP Is the op code of the call instruction used for DOES> L
C;  A synonym for END-CODE

Deferring the definitions of the commas, marks, and resolves
  allows the same assembler to serve for both the system and th
  Meta-Compiler.
```

1 3

```
 0 \ 8086 Assembler   Register Definitions          ks 19 mär 88
 1 ¦ : REG    11 * SWAP 1000 * OR CONSTANT  ;
 2 ¦ : REGS   ( MODE N -- )   SWAP 0 DO  DUP I REG  LOOP  DROP ;
 3
 4 10 0 REGS   AL  CL  DL  BL  AH  CH  DH  BH
 5 10 1 REGS   AX  CX  DX  BX  SP  BP  SI  DI
 6 10 2 REGS   [BX+SI] [BX+DI] [BP+SI] [BP+DI] [SI] [DI] [BP] [BX]
 7  4 2 REGS   [SI+BX] [DI+BX] [SI+BP] [DI+BP]
 8  4 3 REGS   ES  CS  SS  DS
 9  3 4 REGS   #   #)  S#)
10
11 BP Constant UP   [BP] Constant [UP]   \ User Pointer
12 SI CONSTANT IP   [SI] CONSTANT [IP]   ( INTERPRETER POINTER )
13 DI Constant W    [DI] Constant [W]    \ WORKING REGISTER
14 BX Constant RP   [BX] Constant [RP]   \ Return Stack Pointer
15 DX Constant TOS                       \ Top Of Stack im Register
```

18

```
\ 8086 Assembler   Register Definitions              120ct83map

On the 8086, register names are cleverly defined constants.

The value returned by registers and by modes such as #) contains
both mode and register information. The instructions use the
mode information to decide how many arguments exist, and what to
assemble.
  Like many CPUs, the 8086 uses many 3 bit fields in its opcodes
This makes octal ( base 8 ) natural for describing the registers


We redefine the Registers that FORTH uses to implement its
virtual machine.
```

1 4

```
 0 \ Addressing Modes                               ks 19 mär 88
 1 ¦ : MD   CREATE  1000 * ,  DOES> @ SWAP 7000 AND = 0<>  ;
 2 ¦ 0 MD R8?   ¦ 1 MD R16?   ¦ 2 MD MEM?   ¦ 3 MD SEG?  ¦ 4 MD #?
 3 ¦ : REG?   ( n -- f )   7000 AND 2000 < 0<> ;
 4 ¦ : BIG?   ( N -- F )   ABS -200 AND 0<>  ;
 5 ¦ : RLOW   ( n1 -- n2 )    7 AND ;
 6 ¦ : RMID   ( n1 -- n2 )   70 AND ;
 7 ¦ VARIABLE SIZE   SIZE ON
 8 : BYTE   ( -- )   SIZE OFF ;
 9 ¦ : OP,   ( N OP -- )   OR C,  ;
10 ¦ : W,   ( OP MR -- )   R16? 1 AND OP,  ;
11 ¦ : SIZE,   ( OP -- OP' )   SIZE @ 1 AND OP,  ;
12 ¦ : ,/C,  ( n f -- )   IF  ,  ELSE  C,  THEN  ;
13 ¦ : RR,   ( MR1 MR2 -- )   RMID SWAP RLOW OR 300 OP,  ;
14 ¦ VARIABLE LOGICAL
15 ¦ : B/L?   ( n -- f )   BIG? LOGICAL @ OR  ;
```

19

```
\ Addressing Modes                                  160ct83ma
MD  defines words which test for various modes.
R8? R16? MEM? SEG? #?  test for mode equal to 0 thru 4.
REG?  tests for any register mode ( 8 or 16 bit).
BIG?  tests offsets size. True if won't fit in one byte.
RLOW  mask off all but low register field.
RMID  mask off all but middle register field.
SIZE  true for 16 bit, false for 8 bit.
BYTE  set size to 8 bit.
OP,  for efficiency. OR two numbers and assemble.
W,  assemble opcode with W field set for size of register.
SIZE,  assemble opcode with W field set for size of data.
,/C,  assemble either 8 or 16 bits.
RR,  assemble register to register instruction.
LOGICAL  true while assembling logical instructions.
B/L?  see 13MI
```

1 5

```
 0 \ Addressing                                     ks 19 mär 88
 1 ¦ : MEM,   ( DISP MR RMID -- )   OVER #) =
 2     IF  RMID 6 OP, DROP ,
 3     ELSE  RMID OVER RLOW OR -ROT [BP] = OVER 0= AND
 4     IF  SWAP 100 OP, C,  ELSE  SWAP OVER BIG?
 5     IF  200 OP, ,  ELSE  OVER 0=
 6     IF  C, DROP  ELSE  100 OP, C,
 7     THEN THEN THEN THEN  ;
 8 ¦ : WMEM,   ( DISP MEM REG OP -- )   OVER W, MEM,  ;
 9 ¦ : R/M,   ( MR REG -- )
10     OVER REG? IF  RR,  ELSE  MEM,  THEN  ;
11 ¦ : WR/SM,   ( R/M R OP -- )   2 PICK DUP REG?
12     IF  W, RR,  ELSE  DROP SIZE, MEM,  THEN  SIZE ON  ;
13 ¦ VARIABLE INTER
14 : FAR   ( -- )   INTER ON  ;
15 ¦ : ?FAR   ( n1 -- n2 )   INTER @ IF  10 OR  THEN  INTER OFF ;
```

20

```
\ Addressing                                        160ct83ma
These words perform most of the addressing mode encoding.
MEM,  handles memory reference modes.  It takes a displacement,
  a mode/register, and a register, and encodes and assembles
  them.


WMEM,  uses MEM, after packing the register size into the opcod
R/M,  assembles either a register to register or a register to
  or from memory mode.
WR/SM,  assembles either a register mode with size field, or a
  memory mode with size from SIZE. Default is 16 bit. Use BYTE
  for 8 bit size.
INTER  true if inter-segment jump, call, or return.
FAR  sets INTER true.  Usage:  FAR JMP,   FAR CALL,   FAR RET.
?FAR  sets far bit, clears flag.
```

1 6

```
 0 \ Defining Words to Generate Op Codes           ks 19 mär 88
 1 : 1MI   CREATE  C,  DOES>  C@ C,  ;
 2 : 2MI   CREATE  C,  DOES>  C@ C,  12 C,  ;
 3 : 3MI   CREATE  C,  DOES>  C@ C,  HERE - 1-
 4     DUP -200 177 uWITHIN NOT ABORT" Branch out of Range" C, ;
 5 : 4MI   CREATE  C,  DOES>  C@ C,  MEM,  ;
 6 : 5MI   CREATE  C,  DOES>  C@ SIZE,  SIZE ON ;
 7 : 6MI   CREATE  C,  DOES>  C@ SWAP W,  ;
 8 : 7MI   CREATE  C,  DOES>  C@ 366 WR/SM, ;
 9 : 8MI   CREATE  C,  DOES>  C@ SWAP R16? 1 AND OR  SWAP # =
10     IF  C, C,  ELSE  10 OR  C,  THEN  ;
11 : 9MI   CREATE  C,  DOES>  C@  OVER R16?
12     IF  100 OR SWAP RLOW OP,  ELSE  376 WR/SM,  THEN  ;
13 : 10MI  CREATE  C,  DOES>  C@ OVER CL =
14     IF  NIP 322  ELSE  320  THEN  WR/SM,  ;
15
```

21

```
\ Defining Words to Generate Op Codes                120Oct83ma
1MI  define one byte constant instructions.
2MI  define ascii adjust instructions.
3MI  define branch instructions, with one byte offset.
4MI  define LDS, LEA, LES instructions.
5MI  define string instructions.
6MI  define more string instructions.
7MI  define multiply and divide instructions.
8MI  define input and output instructions.

9MI  define increment/decrement instructions.

10MI  define shift/rotate instructions.
*NOTE*  To allow both 'ax shl' and 'ax cl shl', if the register
on top of the stack is cl, shift second register by cl. If not,
shift top ( only) register by one.
```

1 7

```
 0 \ Defining Words to Generate Op Codes           ks 19 mär 88
 1 : 11MI   CREATE  C, C,  DOES>  OVER #) =
 2   IF  NIP C@ INTER @
 3     IF  1 AND IF  352  ELSE  232  THEN  C,  SWAP , ,  INTER OFF
 4     ELSE  SWAP HERE - 2- SWAP  2DUP 1 AND SWAP BIG? NOT AND
 5       IF  2 OP, C,  ELSE  C,  1- ,  THEN  THEN
 6   ELSE  OVER S#) = IF  NIP #) SWAP  THEN
 7     377 C, 1+ C@ ?FAR  R/M,  THEN  ;
 8 : 12MI   CREATE  C, C, C,  DOES>  OVER REG?
 9   IF  C@ SWAP RLOW OP,  ELSE  1+ OVER SEG?
10     IF  C@ RLOW SWAP RMID OP,
11     ELSE  COUNT SWAP C@ C,  MEM,
12   THEN THEN  ;
13 : 14MI   CREATE  C,  DOES> C@
14   DUP ?FAR C,  1 AND 0= IF  ,  THEN ;
15
```

22

```
\ Defining Words to Generate Op Codes                09Apr84ma
11MI  define calls and jumps.
 notice that the first byte stored is E9 for jmp and E8 for cal
so C@ 1 AND  is zero for call, 1 for jmp.
 syntax for direct intersegment:   address segment #) FAR JMP

12MI  define pushes and pops.

14MI  defines returns.
  RET    FAR RET    n +RET   n FAR +RET
```

1 8

```
 0 \ Defining Words to Generate Op Codes           ks 19 mär 88
 1 : 13MI   CREATE  C, C,  DOES>  COUNT >R C@ LOGICAL !  DUP REG?
 2   IF  OVER REG?
 3     IF  R> OVER W, SWAP RR,  ELSE  OVER DUP MEM? SWAP #) = OR
 4     IF  R> 2 OR WMEM,  ELSE  ( # ) NIP  DUP RLOW 0= ( ACC? )
 5     IF  R> 4 OR OVER W, R16? ,/C,
 6     ELSE  OVER B/L? OVER R16? 2DUP AND
 7       -ROT 1 AND SWAP NOT 2 AND OR 200 OP,
 8       SWAP RLOW 300 OR R> OP,  ,/C,
 9     THEN  THEN  THEN
10   ELSE  ( MEM )  ROT DUP REG?
11     IF  R> WMEM,
12     ELSE  ( # ) DROP  2 PICK B/L? DUP NOT 2 AND 200 OR SIZE,
13       -ROT R> MEM,  SIZE @ AND ,/C,  SIZE ON
14   THEN  THEN  ;
15
```

23

```
\ Defining Words to Generate Op Codes                160Oct83ma
13MI  define arithmetic and logical instructions.
```

1 9

```
 0 \ Instructions                                  ks 19 mär 88
 1 : TEST   ( source dest -- )   DUP REG?
 2   IF  OVER REG?
 3     IF  204 OVER W, SWAP RR,  ELSE  OVER DUP MEM? SWAP #) = OR
 4     IF  204 WMEM,  ELSE  ( # ) NIP  DUP RLOW 0= ( ACC? )
 5     IF  250 OVER W,
 6     ELSE  366 OVER W,  DUP RLOW 300 OP,
 7     THEN   R16? ,/C,  THEN  THEN
 8   ELSE  ( MEM )  ROT DUP REG?
 9     IF  204 WMEM,
10     ELSE  ( # ) DROP  366 SIZE,  0 MEM,  SIZE @ ,/C,  SIZE ON
11   THEN  THEN  ;
12
13
14
15
```

24

```
\ Instructions                                    160ct83map
TEST  bits in dest
```

1 10

```
 0 \ Instructions                                  ks 19 mär 88
 1 HEX
 2 : ESC   ( source ext-opcode -- )   RLOW 0D8 OP, R/M,  ;
 3 : INT   ( N -- )   0CD C,  C,  ;
 4 : SEG   ( SEG -- )   RMID 26 OP,  ;
 5 : XCHG   ( MR1 MR2 -- )   DUP REG?
 6   IF  DUP AX =
 7     IF  DROP RLOW 90 OP,  ELSE  OVER AX =
 8     IF  NIP  RLOW 90 OP,  ELSE  86 WR/SM,  THEN  THEN
 9   ELSE  ROT 86 WR/SM,  THEN  ;
10
11 : CS:   CS SEG ;
12 : DS:   DS SEG ;
13 : ES:   ES SEG ;
14 : SS:   SS SEG ;
15
```

25

```
\ Instructions                                    160ct83ma

ESC
INT  assemble interrupt instruction.
SEG  assemble segment instruction.
XCHG  assemble register swap instruction.




CS: DS: ES: SS: assemble segment over-ride instructions.
```

1 11

```
 0 \ Instructions                                  ks 19 mär 88
 1 : MOV   ( S D -- )   DUP SEG?
 2   IF  8E C, R/M,  ELSE  DUP REG?
 3     IF  OVER #) = OVER RLOW 0= AND
 4       IF  A0 SWAP W,   DROP   ,  ELSE  OVER SEG?
 5       IF  SWAP 8C C, RR,  ELSE  OVER # =
 6       IF  NIP DUP R16? SWAP RLOW OVER 8 AND OR B0 OP, ,/C,
 7       ELSE  8A OVER W, R/M,  THEN THEN THEN
 8     ELSE  ( MEM ) ROT DUP SEG?
 9       IF  8C C, MEM,  ELSE  DUP # =
10       IF  DROP C6 SIZE, 0 MEM,  SIZE @ ,/C,
11       ELSE  OVER #) = OVER RLOW 0= AND
12       IF  A2 SWAP W,  DROP   ,   ELSE  88 OVER W, R/M,
13   THEN THEN THEN THEN THEN   SIZE ON  ;
14
15
```

26

```
\ Instructions                                    120ct83ma
MOV  as usual, the move instruction is the most complicated.
 It allows more addressing modes than any other, each of which
 assembles something more or less unique.
```

1 12

```
 0 \ Instructions                                    120ct83map
 1  37  1MI AAA     D5  2MI AAD     D4  2MI AAM     3F  1MI AAS
 2 0 10 13MI ADC  0 00 13MI ADD   2 20 13MI AND  10 E8 11MI CALL
 3  98  1MI CBW     F8  1MI CLC     FC  1MI CLD     FA  1MI CLI
 4  F5  1MI CMC   0 38 13MI CMP     A6  5MI CMPS    99  1MI CWD
 5  27  1MI DAA     2F  1MI DAS     08  9MI DEC     30  7MI DIV
 6        ( ESC )   F4  1MI HLT     38  7MI IDIV    28  7MI IMUL
 7  E4  8MI IN      00  9MI INC         ( INT )  0CE  1MI INTO
 8 0CF  1MI IRET    77  3MI JA      73  3MI JAE     72  3MI JB
 9  76  3MI JBE     E3  3MI JCXZ    74  3MI JE      7F  3MI JG
10  7D  3MI JGE     7C  3MI JL      7E  3MI JLE  20 E9 11MI JMP
11  75  3MI JNE     71  3MI JNO     79  3MI JNS     70  3MI JO
12  7A  3MI JPE     7B  3MI JPO     78  3MI JS      9F  1MI LAHF
13  C5  4MI LDS     8D  4MI LEA     C4  4MI LES     F0  1MI LOCK
14 0AC  6MI LODS    E2  3MI LOOP    E1  3MI LOOPE   E0  3MI LOOPNE
15
```

1 13

```
 0 \ Instructions                                    12Apr84map
 1        ( MOV )   0A4  5MI MOVS    20  7MI MUL     18  7MI NEG
 2  90  1MI NOP      10  7MI NOT   2 08 13MI OR      E6  8MI OUT
 3             8F 07 58 12MI POP     9D  1MI POPF
 4           0FF 36 50 12MI PUSH    9C  1MI PUSHF
 5  10 10MI RCL      18 10MI RCR
 6  F2  1MI REP      F2  1MI REPNZ   F3  1MI REPZ
 7  C3 14MI RET      00 10MI ROL      8 10MI ROR     9E  1MI SAHF
 8  38 10MI SAR    0 18 13MI SBB    0AE  5MI SCAS        ( SEG )
 9  20 10MI SHL      28 10MI SHR     F9  1MI STC     FD  1MI STD
10  FB  1MI STI     0AA  6MI STOS  0 28 13MI SUB         ( TEST )
11  9B  1MI WAIT          ( XCHG )  D7  1MI XLAT  2 30 13MI XOR
12  C2 14MI +RET
13
14
15
```

1 14

```
 0 \ Structured Conditionals                      ks 19 mär 88
 1 : A?>MARK    ( -- f addr ) TRUE   HERE   0 C,   ;
 2 : A?>RESOLVE ( f addr -- ) HERE OVER 1+ - SWAP C! true ?pairs ;
 3 : A?<MARK    ( -- f addr ) TRUE   HERE   ;
 4 : A?<RESOLVE ( f addr -- ) HERE 1+ -  C,  true ?pairs ;
 5 ' A?>MARK    ASSEMBLER IS ?>MARK
 6 ' A?>RESOLVE ASSEMBLER IS ?>RESOLVE
 7 ' A?<MARK    ASSEMBLER IS ?<MARK
 8 ' A?<RESOLVE ASSEMBLER IS ?<RESOLVE
 9 HEX
10 75 CONSTANT 0=   74 CONSTANT 0<>   79 CONSTANT 0<
11 78 CONSTANT 0>=  7D CONSTANT <     7C CONSTANT >=
12 7F CONSTANT <=   7E CONSTANT >     73 CONSTANT U<
13 72 CONSTANT U>=  77 CONSTANT U<=   76 CONSTANT U>
14 71 CONSTANT OV
15 DECIMAL
```

27

```
 \ Instructions                                      120ct83ma
 Most instructions are defined on these two screens. Mnemonics i
 parentheses are defined earlier or not at all.
```

28

```
 \ Instructions                                      120ct83ma
 Most instructions are defined on these two screens. Mnemonics i
 parentheses are defined earlier or not at all.
```

29

```
 \ Structured Conditionals                           160ct83ma
 A?>MARK     assembler version of forward mark.
 A?>RESOLVE  assembler version of forward resolve.
 A?<MARK     assembler version of backward mark.
 A?<RESOLVE  assembler version of backward resolve.
```

These conditional test words leave the opcodes of conditional branches to be used by the structured conditional words.

For example,

```
5 # CX CMP   0< IF  AX BX ADD  ELSE  AX BX SUB  THEN
```

1 0

```
 0 \\ Printer Interface                                  ks 09 mai 88
 1
 2 Dieses File enthaelt das Printer Interface zwischen volksFORTH
 3 und dem Drucker.
 4
 5 Damit ist es moeglich Source-Texte auf bequeme Art und Weise
 6 in uebersichtlicher Form auszudrucken (6 auf eine Seite).
 7
 8 In Verbindung mit dem Multitasker ist es moeglich, auch Texte im
 9 Hintergrund drucken zu lassen und trotztdem weiterzuarbeiten.
10
11 Diese Druckersteurung wurde von U.Hoffmann für das CP/M
12 volksFORTH geschaffen und von K.Schleisiek verändert.
13
14
15
```

1 1

```
 0 \ Printer Interface IBM Graphic Printer          ks 08 aug 88
 1   Onlyforth
 2   Vocabulary Printer   Printer definitions also
 3
 4   Variable pcol   pcol off
 5   Variable prow   prow off
 6   Variable prints prints off
 7
 8   2 &10 thru  .( Interface für IBM Graphic Printer geladen) cr
 9 \ &11 load    .( Spooler geladen) cr
10
11   : plist  ( scr -- )  prints lock  output push
12      print 10cpi cr list cr 5 lf's  prints unlock ;
13
14   Onlyforth
15
```

1 2

```
 0 \ Printer controls                                ks 23 mär 88
 1
 2 ¦ : ctrl:   ( char -- )   Create c,   Does> c@ lst! ;
 3
 4
 5   8   ctrl: ~bs
 6  $D   ctrl: ~cr
 7  $A   ctrl: ~lf
 8  $C   ctrl: ~ff
 9 $1B ¦ ctrl: ESC
10 $12   ctrl: 10cpi
11  $F   ctrl: 17cpi
12
13
14
15
```

6

```
\ Printer output                                    ks 24 mär 88

  : +emit   dup (emit pemit ;
  : +cr     (cr pcr ;
  : +del    (del pdel ;
  : +page   (page ppage ;
  : +at     2dup (at pat ;

¦ Output: >printer   pemit pcr tipp pdel ppage pat pat? ;
¦ Output: +printer  +emit +cr tipp  +del +page +at (at? ;

Forth definitions

: print   >printer normal ;
: +print  +printer normal ;
```

7

```
\ Variables and Setup                               ks 09 mai 88

  Printer definitions

  $00 ¦ Constant logo
  ¦ Variable pageno
  ¦ Create scr#s   &14 allot  \ enough room for 6 screens

¦ : header  ( -- )
     normal 4 spaces dark  ." Seite " pageno @ 2 .r
     &13 spaces  ."  volksFORTH83 der FORTH-Gesellschaft eV "
     5 spaces  file?  -dark 1 pageno +! ~lf ;
```

8

```
\ Print 2 screens across on a page                  ks 03 apr 88

¦ : pr  ( scr# -- )   dup  capacity 1- u>
     IF  drop logo  THEN  1 scr#s +!  scr#s dup @ 2* + ! ;

¦ : 2pr  ( scr#1 scr#2 line# -- )
     cr  17cpi dup 2 .r space  c/l * >r
     pad $101 bl fill  swap block  r@ +  pad c/l cmove
     block r> + pad c/l + 1+ c/l cmove
     pad $101 -trailing type ;

¦ : 2scr  ( scr#1 scr#2 -- )  cr cr normal &17 spaces
     wide dark over 4 .r  &18 spaces  dup 4 .r  -wide -dark
     cr  1/s 0 DO  2dup  I 2pr  LOOP  2drop ;

¦ : pr-start  ( --)   scr#s off  1 pageno ! ;
```

1 3

```
 0 \ printer controls                                  ks 24 mär 88
 1
 2 : #esc:   ( cn..c1 n -- )   Create  dup c, 0 DO c, LOOP
 3   Does> ESC  count bounds DO  I c@ lst!  LOOP ;
 4 $3A 1 #esc: 12cpi
 5
 6 $47 $25 2 #esc: cursive  $48 $25 2 #esc: -cursive
 7 $50 $25 2 #esc: prop     $51 $25 2 #esc: -prop
 8 $33 $49 2 #esc: nlq      $31 $49 2 #esc: standard
 9 $30 $23 2 #esc: fast
10 $31 $57 2 #esc: wide     $30 $57 2 #esc: -wide
11     $47 1 #esc: dark         $48 1 #esc: -dark
12     $32 1 #esc: 6/"          $30 1 #esc: 8/"
13 $31 $2D 2 #esc: +under   $30 $2D 2 #esc: -under
14
15
```

1 4

```
 0 \ printer controls                                  ks 30 apr 88
 1
 2
 3
 4   : <rand  ( +n -- )  ESC $58 lst! lst! &300 lst! ;
 5
 6   : lf's   ( +n -- )  0 DO  ^lf  LOOP ;
 7
 8   : normal     standard 12cpi ^cr ;
 9
10
11
12
13
14
15
```

1 5

```
 0 \ Printer output functions                          ks 07 jan 88
 1
 2   : pemit  ( char -- )    1 pcol +!  dup BL u<
 3     IF  $40 or +under lst! -under exit  THEN  lst! ;
 4
 5   : pcr    ^cr ^lf 1 prow +!   pcol off ;
 6
 7   : pdel   ^bs  pcol @ 1- 0 max pcol ! ;
 8
 9   : ppage  ^ff  prow off   pcol off ;
10
11   : pat    ( row col -- )   dup pcol @ -  dup 0< swap
12     abs 0 DO  BL over IF  drop 8  THEN  lst!  LOOP  drop
13     pcol !   prow ! ;
14
15   : pat?   ( -- row col )   prow @   pcol @ ;
```

9

```
 \ Printer 6 screens on a page                         ks 03 apr 88

 : pagepr      header  scr#s off  scr#s 2+
     3 0 DO  dup @  over 6 + @  2scr  2+  LOOP  drop  page ;

 : shadowpr    header  scr#s off  scr#s 2+
     3 0 DO  dup @  over 2+ @  2scr  4 +  LOOP  drop  page ;

 : pr-flush  ( -- f )  \ any screens left over?
     scr#s @  dup 0=exit  0<>
     BEGIN  scr#s @ 5 < WHILE  -1 pr  REPEAT  logo pr ;

 Variable shadow

 : full?  ( -- f )  scr#s @ 6 = ;
```

10

```
 \ Printer 6 screens on a page                         ks 09 mai 88
 Forth definitions

 : pthru  ( first last -- )  [ Printer ]
    prints lock  output push  print  pr-start  1+ swap
    ?DO  I pr    full? IF  pagepr  THEN  LOOP
    pr-flush IF  pagepr  THEN  prints unlock ;

 : document  ( first last -- ) [ Printer ]
    isfile@ IF capacity 2/ shadow ! THEN
    prints lock  output push  print  pr-start  1+ swap
    ?DO  I pr  I shadow @ + pr   full? IF  shadowpr  THEN  LOOP
    pr-flush  IF  shadowpr  THEN  prints unlock ;

 : listing    0  capacity 2/ 1- document ;
```

11

```
 \ Printerspool                                        ks 30 apr 88

 \needs Task \\

 Input: noinput    0  false  drop  2drop  ;


 noinput  $100 $200 Task spooler  keyboard

 : spool   ( from to -- )
    isfile@  spooler 3 pass  isfile !  pthru  stop ;
```

```
1                    O

 0 \\ Install Editor
 1
 2 Dieses File enthaelt einen Installer fuer den Editor.
 3
 4 Es werden nacheinander die Tasten erfragt, die einen bestimmten
 5 Befehl ausloesen sollen.
 6
 7 Damit ist es moeglich, die Tastatur an die individuellen
 8 Beduerfnisse anzupassen.
 9
10
11
12
13
14
15
```

```
1

\ install Editor                                        UH 12Sep87

Onlyforth Editor also  save   warning on

: tab      &20 col &20 mod - spaces ;
: .key ( c -- )
    dup $7E > IF ." $" u. exit THEN
    dup bl  < IF ." ^" [ Ascii A 1- ] Literal +  THEN emit ;

: install   \ install editor's keyboard
   page ." Entsprechende Tasten druecken. (Blank uebernimmt.)"
   #keys 0 ?DO  cr I 2* actiontable + @ >name .name
     tab ." : " I 2* keytable + dup @ .key   tab ." -> "
     key dup bl = IF drop dup @ THEN  dup .key  swap !
   LOOP ;
-->
```

```
1                    O

 0 \\ Install Editor
 1
 2 Dieses File enthaelt einen Installer fuer den Editor.
 3
 4 Es werden nacheinander die Tasten erfragt, die einen bestimmten
 5 Befehl ausloesen sollen.
 6
 7 Damit ist es moeglich, die Tastatur an die individuellen
 8 Beduerfnisse anzupassen.
 9
10
11
12
13
14
15
```

```
O

\\ Install Editor

Dieses File enthaelt einen Installer fuer den Editor.

Es werden nacheinander die Tasten erfragt, die einen bestimmten
Befehl ausloesen sollen.

Damit ist es moeglich, die Tastatur an die individuellen
Beduerfnisse anzupassen.
```

```
1                    O

 0 \\ Install Editor
 1
 2 Dieses File enthaelt einen Installer fuer den Editor.
 3
 4 Es werden nacheinander die Tasten erfragt, die einen bestimmten
 5 Befehl ausloesen sollen.
 6
 7 Damit ist es moeglich, die Tastatur an die individuellen
 8 Beduerfnisse anzupassen.
 9
10
11
12
13
14
15
```

```
O

\\ Install Editor

Dieses File enthaelt einen Installer fuer den Editor.

Es werden nacheinander die Tasten erfragt, die einen bestimmten
Befehl ausloesen sollen.

Damit ist es moeglich, die Tastatur an die individuellen
Beduerfnisse anzupassen.
```

```
1                    0                                   74

 0 \  ####  volksFORTH   ####                    ks 11 mai 88  \\ -text (find                                ks 02 okt 87
 1 Entwicklung des volksFORTH-83 von
 2                                                              : -text ( adr1 len adr2 -- 0< 1<2 / 0= 1=2 / 0> 1>2 )
 3    K. Schleisiek, B. Pennemann, G. Rehfeld, D. Weineck         over bounds
 4                                                                DO  drop count I c@ - dup IF LEAVE THEN  LOOP nip ;
 5 Zuerst für den 6502 von B.Pennemann und K.Schleisiek
 6 Anpassung für C64 "ultraFORTH"   von G.Rehfeld               : (find    ( string thread -- str false / NFA +n )
 7 Anpassung für 68000   und TOS    von D.Weineck und B.Pennemann   over c@ $1F and >r  @
 8 Anpassung für 8080    und CP/M   von U.Hoffmann    jul 86      BEGIN  dup WHILE  dup @   swap 2+   dup c@ $1F and  r@  =
 9 Anpassung für C16 "ultraFORTH"   von C.Vogt                                  IF  dup 1+  r@  4 pick 1+ -text
10 Anpassung für 8088/86 und MS-DOS von K.Schleisiek  dez 87                       0= IF  rdrop -rot drop exit
11                                                                                THEN   THEN  drop
12 Diese Version 3.80 steht auf den aufgeführten Rechnern in    REPEAT  rdrop ;
13 identischen Versionen zur Verfügung. Das Fileinterface ist
14 unausgereift und wird in der Version 3.90 entscheidend ver-
15 bessert sein.

1                    1                                   75

 0 \ MS-DOS   volksForth Load Screen              ks 03 apr 88  \ find  '  [compile]  [']  nullstring?          ks 29 oct 86
 1   Onlyforth  \needs Transient   include meta.scr
 2                                                              : find    ( string -- acf n / string false )
 3   2 loadfrom META.SCR                                          context   dup @  over 2- @  = IF  2-  THEN
 4                                                                BEGIN  under @ (find  IF  nip found exit  THEN
 5   new FORTH.COM   Onlyforth Target definitions                       swap 2-   dup vp = UNTIL  drop false ;
 6
 7   4 &111 thru          \ Standard 8088-System                : '    ( -- cfa )      name find ?exit Error" ?" ;
 8
 9   flush                 \ close FORTH.COM                    : [compile]       ' , ;                 immediate restrict
10
11 cr .( neuer Kern als FORTH.COM erzeugt) cr bell              : [']             ' [compile] Literal ; immediate restrict
12
13                                                              : nullstring?   ( string -- string false / true )
14                                                                dup c@  0= dup 0=exit  nip ;
15

1                    2                                   76

 0 \\ Die Nutzung der 8088/86 Register            ks 27 oct 86  \ interpreter                                  ks 07 dez 87
 1
 2 Im Assembler sind Forthgemaesse Namen fuer die Register gewaehlt   Defer notfound
 3 Dabei ist die Zuordnung zu den Intel Namen folgendermassen:
 4                                                              : interpreter   ( string -- )   find ?dup
 5 A <=> AX      A- <=> AL     A+ <=> AH                          IF  1 and IF  execute exit  THEN
 6 C <=> CX      C- <=> CL     C+ <=> CH                             Error" compile only"
 7   Register A und C sind zur allgemeinen Benutzung frei         THEN  number? ?exit  notfound ;
 8
 9 D <=> DX      D- <=> DL     D+ <=> DH                        : compiler    ( string -- )   find ?dup
10   das oberste Element des (Daten)-Stacks.                     IF  0> IF  execute exit  THEN   , exit  THEN
11                                                               number? ?dup IF  0> IF  swap [compile] Literal  THEN
12 R <=> BX      R- <=> RL     R+ <=> RH                                          [compile] Literal  exit
13   der Return_stack_pointer                                               THEN  notfound ;
14
15
```

1 3

```
0 \\ Die Nutzung der 8088/86 Register                ks 27 oct 86
1
2 U <=> BP     User_area_pointer
3 S <=> SP     Daten_stack_pointer
4 I <=> SI     Instruction_pointer
5 W <=> DI     Word_pointer, im allgemeinen zur Benutzung frei.
6
7 D: <=> DS    E: <=> ES    S: <=> SS    C: <=> CS
8    Alle Segmentregister werden beim booten auf den Wert des
9    Codesegments C: gesetzt und muessen, wenn sie "verstellt"
10   werden, wieder auf C: zurueckgesetzt werden.
11
12
13
14
15
```

1 4

```
0 \ FORTH Preamble and ID                            ks 11 mär 89
1 Assembler
2
3 nop   5555 # jmp              here 2- >label >cold
4 nop   5555 # jmp              here 2- >label >restart
5
6 Create origin   here origin!    here $100 0 fill
7 \ Hier beginnen die Kaltstartwerte der Benutzervariablen
8
9  $E9 int  end-code  -4 ,  $FC allot
10 \ this is the multitasker initialization in the user area
11
12 ¦ Create logo ," volksFORTH-83 rev. 3.81.41"
13
14
15
```

1 5

```
0 \ Next                                             ks 27 oct 86
1
2  Variable next-link    0 next-link !
3
4  Host Forth Assembler also definitions
5
6  : Next    lods   A W xchg   W ) jmp
7            there tnext-link @ T , H tnext-link ! ;
8
9 \ Next ist in-line code. Fuer den debugger werden daher alle
10 \ "nexts" in einer Liste mit dem Anker NEXT-LINK verbunden.
11
12 : u'       ( -- offset )    T ' 2+ c@ H ;
13
14 Target
15
```

77

```
\ compiler [ ]                                       ks 16 sep 88

: no.extensions  ( string -- )
  state @ IF  Abort" ?"  THEN  Error" ?" ;

' no.extensions Is notfound

Defer parser   ( string -- )    ' interpreter Is parser

: interpret
  BEGIN  ?stack name nullstring? IF  aborted off exit  THEN
         parser  REPEAT ;

: [      ['] interpreter Is parser  state off ; immediate

: ]      ['] compiler    Is parser  state on ;
```

78

```
\  Is                                                ks 07 dez 87

  : (is      r> dup 2+ >r @ ! ;

¦ : def?  ( cfa -- )
    @  [ ' notfound @   ] Literal   - Abort" not deferred" ;

  : Is   ( addr -- )     ' dup def?   >body
    state @ IF  compile (is , exit  THEN  ! ; immediate
```

79

```
\ ?stack                                             ks 01 okt 87

¦ : stackfull ( -- )     depth $20 > Abort" tight stack"
    reveal last? IF dup heap? IF name> ELSE 4- THEN (forget THE
    true Abort" dictionary full" ;

  Code ?stack    u' dp U D) A mov   S A sub  CS
    ?[ $100 # A add  CS ?[ ;c: stackfull ; Assembler  ]? ]?
    u' s0 U D) A mov   A inc   A inc   S A sub
    CS not ?[  Next  ]?  ;c: true Abort" stack empty" ;

\ : ?stack     sp@ here - $100 u< IF  stackfull  THEN
\              sp@ s0 @ u> Abort" stack empty" ;
```

1 6

```
 0 \ recover ;c: noop                                  ks 27 oct 86
 1
 2  Create recover   Assembler
 3     R dec   R dec   I R ) mov   I pop   Next
 4  end-code
 5
 6 Host Forth Assembler also definitions
 7
 8  :  ;c:   0 T recover # call ] end-code H ;
 9
10 Target
11
12 ¦ Code di    cli              Next   end-code
13 ¦ Code ei    sti     here     Next   end-code
14
15  Code noop       here 2- !   end-code
```

1 7

```
 0 \ User variables                                    ks 16 sep 88
 1  8 uallot drop  \ Platz fuer Multitasker
 2       \ Felder: entry  link  spare  SPsave
 3       \ Laenge kompatibel zum 68000, 6502 und 8080 volksFORTH
 4  User s0
 5  User r0
 6  User dp
 7  User offset           0 offset !
 8  User base             &10 base !
 9  User output
10  User input
11  User errorhandler   \ pointer for Abort" -code
12  User aborted        \ code address of latest error
13  User voc-link
14  User file-link   cr .( Wieso ist UDP Uservariable? )
15  User udp            \ points to next free addr in User_area
```

1 8

```
 0 \ manipulate system pointers                        ks 03 aug 87
 1
 2  Code sp@ ( -- addr )   D push   S D mov   Next   end-code
 3
 4  Code sp! ( addr -- )   D S mov   D pop   Next   end-code
 5
 6
 7  Code up@ ( -- addr )   D push   U D mov   Next   end-code
 8
 9  Code up! ( addr -- )   D U mov   D pop    Next   end-code
10
11  Code ds@ ( -- addr )   D push   D: D mov   Next   end-code
12
13  $10 Constant b/seg    \ bytes per segment
14
15
```

80

```
 \ .status push load                                   ks 29 oct 86

 ¦ Create: pull  r> r> ! ;
   : push   ( addr -- )
     r> swap dup >r @ >r pull >r >r ; restrict

   Defer .status   ' noop Is .status

   : (load  ( blk offset -- )   isfile@ >r
     loadfile @ >r   fromfile @ >r   blk @ >r   >in @ >r
     >in !   blk !  isfile@ loadfile !  .status  interpret
     r> >in !   r> blk !   r> fromfile !   r> loadfile !
     r> isfile ! ;

   : load   ( blk -- )     ?dup 0=exit  0 (load ;
```

81

```
 \ +load thru +thru --> rdepth depth                   ks 26 jul 87

   : +load    ( offset -- )       blk @ + load ;

   : thru     ( from to -- )      1+ swap DO I  load LOOP ;

   : +thru    ( off0 off1 -- )    1+ swap DO I +load LOOP ;

   : -->        1 blk +! >in off .status ; immediate

   : rdepth   ( -- +n )           r0 @ rp@ 2+   - 2/ ;

   : depth    ( -- +n )           sp@ s0 @ swap - 2/ ;
```

82

```
 \  prompt  quit                                       ks 16 sep 88

   : (prompt   .status  state @ IF  cr ." ] " exit  THEN
     aborted @ 0= IF  ." ok"  THEN  cr ;

   Defer prompt    ' (prompt Is prompt

   : (quit  BEGIN  prompt query interpret  REPEAT ;

   Defer 'quit     ' (quit Is 'quit

   : quit     r0 @ rp!   [compile] [   blk off   'quit ;

 \ : classical   cr .status  state @
 \    IF  ." C> " exit  THEN  ." I> " ;
```

1 9

```
0 \ manipulate returnstack                          ks 27 oct 86
1
2  Code rp@ ( -- addr )   D push   R D mov   Next   end-code
3
4  Code rp! ( addr -- )   D R mov   D pop    Next   end-code
5
6
7  Code >r  ( 16b -- )  R dec   R dec   D R ) mov   D pop   Next
8  end-code restrict
9
10 Code r>  ( -- 16b )  D push   R ) D mov   R inc   R inc   Next
11 end-code restrict
12
13
14
15
```

83

```
\ end-trace abort                                   ks 26 jul 87

: standardi/o     [ output ] Literal output 4 cmove ;

Code end-trace    next-link # W mov   $AD # A- mov
   $FF97 # C mov  [[  W ) W mov   W W or  0= not
                  ?[[  A- -4 W D) mov   C -3 W D) mov
                  ]]?  lods   A W xchg   W ) jmp   end-code

Defer 'abort      ' noop Is 'abort

: abort     end-trace clearstack 'abort standardi/o quit ;
```

1 10

```
0 \ r@ rdrop  exit unnest ?exit                     ks 27 oct 86
1  Code r@ ( -- 16b )   D push   R ) D mov   Next   end-code
2
3  Code rdrop          R inc   R inc   Next   end-code restrict
4
5  Code exit
6  Label >exit   R ) I mov   R inc   R inc   Next   end-code
7
8  Code unnest   >exit  here 2- !   end-code
9
10 Code ?exit  ( flag -- )
11    D D or   D pop   >exit 0= ?]      [[  Next   end-code
12
13 Code 0=exit ( flag -- )
14    D D or   D pop   >exit 0= not ?]  ]]   end-code
15 \ : ?exit ( flag -- )   IF rdrop THEN ;
```

84

```
\ (error Abort" Error"                              ks 16 sep 88
  Variable scr      1 scr !
  Variable r#       r# off

  : (error ( string -- )   rdrop r> aborted !  standardi/o
     space here .name   count type space ?cr
     blk @ ?dup IF  scr ! >in @ r# !  THEN  quit ;
  ' (error errorhandler !

  : (abort"    "lit swap IF  >r clearstack r>
     errorhandler perform exit THEN drop ; restrict

¦ : (error"    "lit swap IF  errorhandler perform exit  THEN
               drop ; restrict
```

1 11

```
0 \ execute  perform                                ks 27 oct 86
1
2  Code execute ( acf -- )   D W mov   D pop   W ) jmp   end-code
3
4  Code perform ( addr -- )  D W mov  D pop   W ) W mov   W ) jmp
5  end-code
6
7 \ : perform   ( addr -- )      @ execute ;
8
9
10
11
12
13
14
15
```

85

```
\ -trailing space spaces                            ks 16 sep 88

: Abort"     compile (abort" ," align ; immediate restrict
: Error"     compile (error" ," align ; immediate restrict

$20 Constant bl

: -trailing ( addr n1 -- addr n2)
   dup 0 ?DO  2dup + 1- c@ bl - IF LEAVE THEN  1-  LOOP ;

: space               bl emit ;
: spaces   ( u -- )   0 ?DO  space  LOOP ;
```

1 12

```
0 \ c@ c! ctoggle                                   ks 27 oct 86
1
2  Code c@   ( addr -- 8b )
3     D W mov   W ) D- mov   0 # D+ mov   Next   end-code
4
5  Code c!   ( 16b addr -- )
6     D W mov   A pop   A- W ) mov   D pop   Next   end-code
7
8  Code ctoggle   ( 8b addr -- )
9     D W mov   A pop   A- W ) xor   D pop   Next   end-code
10
11 \ : ctoggle   ( 8b addr -- )   under c@ xor swap c! ;
12
13 Code flip ( 16b1 -- 16b2 )   D- D+ xchg   Next   end-code
14
15
```

1 86

```
\ hold <# #> sign # #s                              ks 29 dez 87

: hld   ( -- addr)          pad 2- ;

: hold   ( char -- )        -1 hld +!   hld @ c! ;

: <#                        hld hld ! ;

: #>     ( 32b -- addr +n )  2drop   hld @   hld over - ;

: sign   ( n -- )           0< not ?exit  Ascii - hold ;

: #      ( +d1 -- +d2)
   base @ ud/mod   rot dup 9 > 7 and +  Ascii 0 +  hold ;

: #s     ( +d -- 0 0 )      BEGIN # 2dup d0= UNTIL ;
```

1 13

```
0 \ @ ! 2@ 2!                                       ks 27 oct 86
1
2  Code @  ( addr -- 16b )  D W mov   W ) D mov   Next   end-code
3
4  Code !  ( 16b addr -- )  D W mov   W ) pop   D pop   Next
5  end-code
6
7  : 2@   ( addr -- 32b )    dup 2+ @   swap @ ;
8
9  : 2!   ( 32b addr -- )    under !   2+ ! ;
10
11
12
13
14
15
```

1 87

```
\ print numbers .s                                  ks 07 feb 89

: d.r   ( d +n -- )   -rot under dabs <# #s rot sign #>
                      rot over max over - spaces type ;
: d.    ( d -- )      0 d.r space ;

: .r    ( n +n -- )   swap extend rot d.r ;
: .     ( n -- )      extend d. ;

: u.r   ( u +n -- )   0 swap d.r ;
: u.    ( u -- )      0 d. ;

: .s    sp@ s0 @ over - $20 umin bounds ?DO I @ u. 2 +LOOP ;
```

1 14

```
0 \ +! drop swap                                    ks 27 oct 86
1
2  Code +!     ( 16b addr -- )
3     D W mov   A pop   A W ) add   D pop   Next   end-code
4
5  \  : +!       ( n addr -- )   under @ + swap ! ;
6
7
8  Code drop   ( 16b -- )   D pop   Next   end-code
9
10 Code swap   ( 16b1 16b2 -- 16b2 16b1 )
11    A pop   D push   A D xchg   Next   end-code
12
13
14
15
```

```
\ list c/l l/s                                      ks 19 mär 88

  &64 Constant c/l        \ Screen line length
  &16 Constant l/s        \ lines per screen

  : list ( scr -- )  dup capacity u<
     IF  scr !  ." Scr " scr @ .
         ." Dr " drv .  isfile@ .file
         l/s 0 DO  cr I 2 .r space   scr @ block
                   I c/l * +   c/l -trailing type
             LOOP  cr exit
     THEN  9 ?diskerror ;
```

1 15

```
0 \ dup  ?dup                                   ks 27 oct 86
1
2  Code dup    ( 16b -- 16b 16b )  D push   Next   end-code
3
4 \ : dup      ( 16b -- 16b 16b )    sp@ @ ;
5
6  Code ?dup   ( 16b -- 16b 16b / false )
7     D D or   0= not ?[  D push  ]?  Next   end-code
8
9 \ : ?dup     ( 16b -- 16b 16b / false)   dup 0=exit dup ;
10
11
12
13
14
15
```

1 16

```
0 \ over rot nip under                          ks 27 oct 86
1
2  Code over   ( 16b1 16b2 -- 16b1 16b2 16b1 )
3     A D xchg   D pop   D push   A push   Next   end-code
4 \ : over  >r dup r> swap ;
5
6  Code rot    ( 16b1 16b2 16b3 -- 16b2 16b3 16b1 )
7     A D xchg  C pop   D pop   C push   A push   Next  end-code
8 \ : rot   >r swap r> swap ;
9
10  Code nip ( 16b1 16b2 -- 16b2 )  S inc  S inc   Next  end-code
11 \ : nip   swap drop ;
12
13  Code under ( 16b1 16b2 -- 16b2 16b1 16b2 )
14     A pop   D push   A push   Next   end-code
15 \ : under swap over ;
```

1 17

```
0 \ -rot pick                                   ks 27 oct 86
1
2  Code -rot    ( 16b1 16b2 16b3 -- 16b3 16b1 16b2 )
3     A D xchg   D pop   C pop   A push   C push   Next  end-code
4
5 \ : -rot    ( 16b1 16b2 16b3 -- 16b3 16b1 16b2 )   rot rot ;
6
7  Code pick    ( n -- 16b.n )
8     D sal   D W mov   S W add   W ) D mov   Next   end-code
9
10 \ : pick    ( n -- 16b.n )     1+ 2* sp@ + @ ;
11
12
13
14
15
```

89

```
\ multitasker primitives                        ks 29 oct 86

Code pause    D push   I push   R push
   S 6 U D) mov   2 U D) U add   4 # U add   U jmp
end-code

: lock ( addr -- )
   dup @  up@  = IF  drop exit  THEN
   BEGIN  dup @ WHILE  pause  REPEAT  up@ swap ! ;

: unlock   ( addr -- )        dup lock off ;

Label wake   Assembler   U pop   2 # U sub   A pop
   popf   6 U D) S mov   R pop   I pop   D pop   Next
end-code
$E9 4 * >label >taskINT
```

90

```
\\ Struktur der Blockpuffer                     ks 04 jul 87

 0 : link zum naechsten Puffer
 2 : file     0 = direct access
             -1 = leer,
             sonst adresse eines file control blocks
 4 : blocknummer
 6 : statusflags   Vorzeichenbit kennzeichnet update
 8 : Data ... 1 Kb ...
```

91

```
\ buffer mechanism                              ks 04 okt 87

Variable isfile      isfile off   \ addr of file control bloc
Variable fromfile    fromfile off \ fcb in kopieroperationen

Variable prev        prev off     \ Listhead
Variable buffers     buffers off  \ Semaphor

$408 Constant b/buf               \ physikalische Groesse
$400 Constant b/blk               \ bytes/block

Defer r/w                         \ physikalischer Diskzugrif
Variable error#      error# off   \ Nummer des letzten Fehler
Defer ?diskerror                  \ Fehlerbehandlung
```

1 18

```
 0 \ roll -roll                                  ks 27 oct 86
 1
 2  Code roll  ( n -- )
 3     A I xchg   D sal   D C mov   D I mov   S I add
 4     I ) D mov   I W mov   I dec   W inc   std
 5     rep byte movs   cld   A I xchg   S inc   S inc   Next
 6  end-code
 7 \ : roll   ( n -- )
 8 \    dup >r  pick sp@ dup 2+  r> 1+ 2* cmove> drop ;
 9
10  Code -roll ( n -- )   A I xchg   D sal   D C mov
11     S W mov   D pop   S I mov   S dec   S dec
12     rep byte movs   D W ) mov   D pop   A I xchg   Next
13  end-code
14 \ : -roll   ( n -- ) >r dup sp@ dup 2+
15 \    dup 2+ swap r@ 2* cmove r> 1+ 2* + ! ;
```

1 92

```
\ (core?                                        ks 28 mai 87

  Code (core? ( blk file -- dataaddr / blk file )
     A pop   A push   D D or   0= ?[  u' offset U D) A add  ]?
     prev #) W mov   2 W D) D cmp   0=
     ?[  4 W D) A cmp   0=
        ?[  8 W D) D lea   A pop   ' exit @ # jmp  ]? ]?
     [[ [[  W ) C mov   C C or   0= ?[  Next  ]?
           C W xchg   4 W D) A cmp   0= ?]  2 W D) D cmp   0= ?]
     W ) A mov   prev #) D mov   D W ) mov   W prev #) mov
     8 W D) D lea   C W mov   A W ) mov   A pop
     ' exit @ # jmp
  end-code
```

1 19

```
 0 \ 2swap  2drop  2dup 2over                    ks 27 oct 86
 1  Code 2swap ( 32b1 32b2 -- 32b2 32b1 )   C pop   A pop   W pop
 2     C push   D push   W push   A D xchg   Next   end-code
 3 \ : 2swap ( 32b1 32b2 -- 32b2 32b1 ) rot >r rot r> ;
 4
 5  Code 2drop ( 32b -- )  S inc   S inc   D pop   Next  end-code
 6 \ : 2drop ( 32b -- ) drop drop ;
 7
 8  Code 2dup ( 32b -- 32b 32b )
 9     S W mov   D push   W ) push   Next   end-code
10 \ : 2dup ( 32b -- 32b 32b ) over over ;
11
12  Code 2over  ( 1 2 x x -- 1 2 x x 1 2 )
13     D push   S W mov   6 W D) push   4 W D) D mov   Next
14  end-code
15 \ : 2over     ( 1 2 x x -- 1 2 x x 1 2 )  3 pick  3 pick ;
```

1 93

```
\\ (core?                                       ks 31 oct 8

 : this? ( blk file bufadr -- flag )
     dup 4+ @  swap 2+ @  d= ;

  .( (core?:  offset is handled differently in code! )

 : (core? ( blk file -- dataaddr / blk file )
     BEGIN  over offset @ + over prev @ this?
        IF  rdrop 2drop prev @ 8 + exit  THEN
        2dup >r offset @ + >r prev @
        BEGIN dup @ ?dup 0= IF  rdrop rdrop drop exit  THEN
              dup r> r> 2dup >r >r rot this?  0=
        WHILE nip REPEAT
        dup @ rot ! prev @ over ! prev ! rdrop rdrop
     REPEAT ;
```

1 20

```
 0 \ and or xor not                              ks 27 oct 86
 1
 2  Code not   ( 16b1 -- 16b2 )  D com   Next   end-code
 3
 4  Code and   ( 16b1 16b2 -- 16b3 )
 5     A pop   A D and   Next   end-code
 6
 7  Code or    ( 16b1 16b2 -- 16b3 )
 8     A pop   A D or   Next   end-code
 9 \ : or      ( 16b1 16b2 -- 16b3 )   not swap not and not ;
10
11  Code xor   ( 16b1 16b2 -- 16b3 )
12     A pop   A D xor   Next   end-code
13
14
15
```

94

```
\ backup emptybuf readblk                       ks 23 jul 87

 : backup ( bufaddr -- )      dup 6+ @ 0<
     IF  2+ dup @ 1+         \ buffer empty if file = -1
         IF  BEGIN  dup 6+ over 2+ @ 2 pick @ 0 r/w
             WHILE  1 ?diskerror  REPEAT
         THEN  4+ dup @ $7FFF and over !  THEN
     drop ;

  : emptybuf ( bufaddr -- )      2+ dup on 4+ off ;

 : readblk ( blk file addr -- blk file addr )
     dup emptybuf  >r
     BEGIN  2dup   0= offset @ and  +
            over   r@ 8 + -rot 1 r/w
     WHILE  2 ?diskerror  REPEAT r>  ;
```

1 21

```
0 \ + -  negate                                  ks 27 oct 86
1
2  Code +   ( n1 n2 -- n3 )   A pop   A D add   Next   end-code
3
4  Code negate  ( n1 -- n2 )    D neg   Next  end-code
5 \ : negate     ( n1 -- n2 )    not 1+ ;
6
7  Code -    ( n1 n2 -- n3 )
8     A pop   D A sub   A D xchg   Next    end-code
9 \ : -    ( n1 n2 -- n3 )   negate + ;
10
11
12
13
14
15
```

1 22

```
0 \ dnegate d+                                   ks 27 oct 86
1
2  Code dnegate ( d1 -- -d1 )     D com   A pop   A neg
3     CS not ?[  D inc  ]?   A push   Next   end-code
4
5  Code d+      ( d1 d2 -- d3 )   A pop   C pop   W pop
6     W A add   A push   C D adc   Next   end-code
7
8
9
10
11
12
13
14
15
```

1 23

```
0 \ 1+ 2+ 3+ 4+ 6+    1- 2- 4-                   ks 27 oct 86
1
2  Code 1+ ( n1 -- n2 )    [[   D inc   Next
3  Code 2+ ( n1 -- n2 )    [[   D inc   swap ]]
4  Code 3+ ( n1 -- n2 )    [[   D inc   swap ]]
5  Code 4+ ( n1 -- n2 )    [[   D inc   swap ]]
6 ¦ Code 6+ ( n1 -- n2 )    D inc   D inc   ]]   end-code
7
8  Code 1- ( n1 -- n2 )    [[   D dec   Next
9  Code 2- ( n1 -- n2 )    [[   D dec   swap ]]
10 Code 4- ( n1 -- n2 )    D dec   D dec   ]]   end-code
11
12
13
14
15
```

95

```
\ take mark updates? full? core?                 ks 04 jul 87
¦ : take ( -- bufaddr)    prev
     BEGIN  dup @ WHILE  @ dup 2+ @ -1 = UNTIL
     buffers lock   dup backup ;

¦ : mark ( blk file bufaddr -- blk file )   2+ >r
     2dup r@ !  over 0= offset @ and +   r@ 2+ !
     r> 4+ off   buffers unlock ;

¦ : updates? ( -- bufaddr / flag)
     prev  BEGIN  @ dup  WHILE  dup 6+ @ 0< UNTIL ;

  : core? ( blk file -- addr /false )   (core? 2drop false ;
```

96

```
\ block & buffer manipulation                    ks 01 okt 87
  : (buffer ( blk file -- addr )
     BEGIN  (core? take mark  REPEAT ;

  : (block ( blk file -- addr )
     BEGIN  (core? take readblk mark  REPEAT ;

  Code isfile@  ( -- addr )
     D push   isfile #) D mov   Next   end-code
\ : isfile@ ( -- addr )    isfile @ ;

  : buffer  ( blk -- addr )   isfile@ (buffer ;

  : block   ( blk -- addr )   isfile@ (block ;
```

97

```
\ block & buffer manipulation                    ks 02 okt 87
  : update          $80 prev @ 6+ 1+ ( Byte-Order! )  c! ;

  : save-buffers    buffers lock
     BEGIN  updates? ?dup WHILE  backup REPEAT  buffers unlock

  : empty-buffers   buffers lock prev
     BEGIN  @ ?dup WHILE  dup emptybuf  REPEAT  buffers unlock

  : flush   file-link
     BEGIN  @ ?dup WHILE  dup fclose  REPEAT
     save-buffers empty-buffers ;
```

1 24

```
 0 \ number Constants                                  ks 30 jan 88
 1 -1 Constant true      0 Constant false
 2
 3       0 ( --  0 )   Constant   0
 4       1 ( --  1 )   Constant   1
 5       2 ( --  2 )   Constant   2
 6       3 ( --  3 )   Constant   3
 7       4 ( --  4 )   Constant   4
 8      -1 ( -- -1 )   Constant  -1
 9
10   Code on  ( addr -- )   -1 # A mov
11 [[   D W mov   A W ) mov   D pop   Next
12   Code off ( addr -- )    0 # A mov   ]]   end-code
13
14 \ : on   ( addr -- )   true  swap ! ;
15 \ : off  ( addr -- )   false swap ! ;
```

1 25

```
 0 \ words for number literals                         ks 27 oct 86
 1
 2   Code lit    ( -- 16b )   D push   I ) D mov   I inc
 3 [[   I inc   Next   end-code restrict
 4
 5   Code clit   ( -- 8b )
 6      D push   I ) D- mov   0 # D+ mov   ]]   end-code restrict
 7
 8   : Literal  ( 16b -- )
 9      dup $FF00 and   IF  compile lit , exit  THEN
10      compile clit c, ; immediate restrict
11
12
13
14
15
```

1 26

```
 0 \ comparision code words                            ks 27 oct 86
 1
 2   Code 0=    ( 16b -- flag )
 3      D D or   0 # D mov   0= ?[  D dec  ]?  Next   end-code
 4
 5   Code 0<>  ( n -- flag )
 6      D D or   0 # D mov   0= not ?[  D dec  ]?  Next   end-code
 7 \ : 0<> ( n -- flag )       0= not ;
 8
 9   Code u<    ( u1 u2 -- flag )   A pop
10 [[   D A sub   0 # D mov  CS ?[  D dec  ]?  Next   end-code
11
12   Code u>    ( u1 u2 -- flag )   A D xchg   D pop  ]]  end-code
13 \ : u>  ( u1 u2 -- flag )    swap u< ;
14
15
```

98

```
 \ Allocating buffers                                  ks 31 oct 86
   $10000 Constant limit     Variable first

   : allotbuffer ( -- )
      first @  r0 @  -  b/buf 2+  u< ?exit
      b/buf negate first +!  first @ dup emptybuf
      prev @ over !  prev ! ;

   : freebuffer ( -- )   first @ limit b/buf - u<
      IF first @  backup  prev
        BEGIN dup @  first @ -  WHILE  @  REPEAT
      first @  @ swap !  b/buf first +!  THEN ;

   : all-buffers  BEGIN  first @ allotbuffer first @ =  UNTIL ;

 ¦ : init-buffers    prev off  limit first !  all-buffers ;
```

99

```
 \ endpoints of forget                                 uh 27 apr 88

 ¦ : ¦? ( nfa -- flag )   c@ $20 and ;

 ¦ : forget? ( adr nfa -- flag )   \ code in heap or above adr ?
      name> under  1+ u<  swap  heap?  or ;

 ¦ : endpoint ( addr sym thread -- addr sym' )
      BEGIN  BEGIN  @  2 pick  over  u> IF  drop exit  THEN
                    dup heap? UNTIL  dup >r 2+ dup ¦?
        IF  >r over r@ forget? IF  r@ (name> >body  umax  THEN
            rdrop  THEN  r>
      REPEAT ;

 ¦ : endpoints ( addr -- addr symb )   heap  voc-link @
      BEGIN  @ ?dup WHILE  dup >r 4- endpoint r> REPEAT ;
```

100

```
 \ remove, -words, -tasks                              ks 30 apr 88
   : remove ( dic sym thread -- dic sym )
      BEGIN dup @ ?dup      \ unlink forg. words
      WHILE dup heap?
        IF  2 pick over u>  ELSE  3 pick over 1+ u<  THEN
        IF  @ over ! ( unlink word)  ELSE nip THEN  REPEAT drop ;

 ¦ : remove-words ( dic sym -- dic sym )   voc-link
      BEGIN  @ ?dup WHILE  dup >r  4- remove  r> REPEAT ;

 ¦ : >up   2+ dup @ 2+ + ;

 ¦ : remove-tasks  ( dic -- )  up@
      BEGIN  dup >up up@ - WHILE  2dup >up swap here uwithin
        IF dup >up >up over - 2- 2- over 2+ !  ELSE  >up  THEN
      REPEAT  2drop ;
```

1 27

```
 0 \  comparision words                          ks 13 sep 88
 1   Code <     ( n1 n2 -- flag )   A pop
 2 [[ [[   D A sub   0 # D mov   < ?[  D dec  ]?  Next   end-code
 3
 4   Code >    ( n1 n2 -- flag )   A D xchg   D pop  ]]  end-code
 5
 6   Code 0>   ( n -- flag )       A A xor          ]]  end-code
 7
 8 \ : <   ( n1 n2 -- flag )
 9 \    2dup xor 0< IF  drop 0< exit  THEN  - 0< ;
10 \ : >   ( n1 n2 -- flag )    swap < ;
11 \ : 0>  ( n -- flag )        negate 0< ;
12
13   Code 0<   ( n1 n2 -- flag )
14      D D or   0 # D mov   0< ?[  D dec  ]?   Next   end-code
15 \ : 0<  ( n1 -- flag )       8000 and 0<> ;
```

1 28

```
 0 \ comparision words                           ks 27 oct 86
 1
 2   Code =    ( n1 n2 -- flag )   A pop   A D cmp
 3      0 # D mov  0= ?[  D dec  ]?   Next   end-code
 4 \ : =   ( n1 n2 -- flag )    - 0= ;
 5
 6   Code uwithin  ( u1 [low high[  -- flag )   A pop   C pop
 7      A C cmp  CS ?[ [[ swap   0 # D mov   Next  ]?
 8           D C cmp  CS ?]  -1 # D mov   Next   end-code
 9 \ : uwithin  ( u1 [low up[  -- f )   over - -rot   - u> ;
10
11   Code case?  ( 16b1 16b2 -- 16b1 ff / tf )  A pop   A D sub
12      0= ?[  D dec  ][  A push   D D xor  ]?  Next   end-code
13 \ : case? ( 16b1 16b2 -- 16b1 false / true )
14 \    over = dup 0=exit  nip ;
15
```

1 29

```
 0 \ double number comparisons                   ks 27 oct 86
 1
 2   Code d0=  ( d - f)      A pop   A D or
 3      0= not ?[  1 # D mov  ]?  D dec   Next   end-code
 4 \ : d0= ( d -- flag )        or 0= ;
 5
 6   : d=  ( d1 d2 -- flag )    dnegate d+ d0= ;
 7
 8 Code d<    ( d1 d2 -- flag )    C pop   A pop
 9    D A sub   A pop   -1 # D mov  < ?[  [[ swap   Next  ]?
10    0= ?[  C A sub  CS ?[  D dec  ]? ]?  D inc   ]]   end-code
11 \ : d<  ( d1 d2 -- flag )
12 \    rot 2dup -  IF  > nip nip exit  THEN  2drop u< ;
13
14
15
```

101

```
 \ remove-vocs trim                              ks 31 oct 86
 : remove-vocs ( dic symb -- dic symb )
    voc-link remove     thru.vocstack
    DO  2dup I @ -rot uwithin
        IF  [ ' Forth 2+ ] Literal I !  THEN  -2 +LOOP
    2dup  current @  -rot  uwithin 0=exit
    [ ' Forth 2+ ] Literal current ! ;

 Defer custom-remove     ' noop Is custom-remove

 : trim   ( dic symb -- )  next-link remove
    over  remove-tasks remove-vocs remove-words remove-files
    custom-remove  heap swap - hallot dp !  last off ;
```

102

```
 \ deleting words from dict.                     ks 02 okt 87

 : clear       here  dup up@  trim  dp ! ;

 : (forget ( adr -- )
    dup heap? Abort" is symbol"  endpoints  trim ;

 : forget   ' dup [ dp ] Literal @  u< Abort" protected"
    >name  dup  heap? IF  name>  ELSE  4-  THEN  (forget ;

 : empty   [ dp ] Literal @ up@ trim
           [ udp ] Literal @ udp ! ;
```

103

```
 \ save bye stop? ?cr                            ks 1UH 26sep8
 : save    here  up@ trim   up@ origin $100 cmove
    voc-link @ BEGIN  dup 4- @ over 2- ! @ ?dup  0= UNTIL ;

 $1B Constant #esc

 : end?   key #esc case? 0=
    IF  #cr case? 0= IF 3 ( Ctrl-C ) - ?exit THEN  THEN
    true rdrop ;

 : stop? ( -- flag )   key? IF  end? end?  THEN  false ;

 : ?cr       col c/l u> 0=exit  cr ;
```

1 30 104

```
 0 \ min max umax umin abs dabs extend              ks 27 oct 86
 1   Code min  ( n1 n2 -- n3 )  A pop   A D sub  < ?[  D A add  ]?
 2                      [[ [[ [[   A D xchg   Next   end-code
 3   Code max  ( n1 n2 -- n3 )
 4      A pop   A D sub  dup < not ?]  D A add    ]]   end-code
 5   Code umin ( u1 u2 -- u3 )
 6      A pop   A D sub  dup CS ?]  D A add       ]]   end-code
 7   Code umax ( u1 u2 -- u3 )
 8      A pop   A D sub  dup CS not ?]  D A add   ]]   end-code
 9
10   Code extend ( n -- d )
11      A D xchg   cwd   A push   Next   end-code
12
13   Code abs ( n -- u )   D D or  0< ?[  D neg  ]?  Next  end-code
14
15   : dabs  ( d -- ud )      extend 0=exit  dnegate ;
```

```
\ in/output structure                          ks 31 oct 86

: Out:   Create dup c, 2+ Does> c@ output @ + perform ;

  : Output:  Create: Does> output ! ;
0  Out: emit   Out: cr   Out: type   Out: del
   Out: page   Out: at   Out: at?    drop

  : row ( -- row )     at? drop ;
  : col ( -- col )     at? nip ;

: In:    Create dup c, 2+ Does> c@ input @ + perform ;

  : Input:   Create:  Does> input ! ;
0  In: key   In: key?   In: decode   In: expect  drop
```

1 31 105

```
 0 \\ min max umax umin extend                    10Mar8
 1
 2 : minimax  ( n1 n2 flag -- n3 )   rdrop IF swap THEN drop ;
 3
 4 : min  ( n1 n2 -- n3 )            2dup  > minimax ;
 5 : max  ( n1 n2 -- n3 )            2dup  < minimax ;
 6 : umax  ( u1 u2 -- u3 )           2dup u< minimax ;
 7 : umin  ( u1 u2 -- u3 )           2dup u> minimax ;
 8 : extend   ( n -- d )             dup 0< ;
 9 : dabs  ( d -- ud )               extend IF dnegate THEN ;
10 : abs   ( n -- u)                 extend IF  negate THEN ;
11
12
13
14
15
```

```
\ Alias  only definitionen                     ks 31 oct 86

   Root definitions

   : seal  [ ' Root >body ] Literal off ; \ "erases" Root Vocab.

   ' Only        Alias Only
   ' Forth       Alias Forth
   ' words       Alias words
   ' also        Alias also
   ' definitions Alias definitions

   Forth definitions
```

1 32 106

```
 0 \ (do (?do endloop  bounds                     ks 30 jan 88
 1
 2   Code (do  ( limit start -- )   A pop
 3 [[   $80 # A+ xor   R dec   R dec   I inc   I inc
 4      I R ) mov   R dec   R dec   A R ) mov   R dec   R dec
 5      A D sub   D R ) mov   D pop   Next  end-code  restrict
 6
 7   Code (?do ( limit start -- )   A pop   A D cmp  0= ?]
 8      I ) I add   D pop   Next   end-code  restrict
 9
10   Code endloop    6 # R add   Next   end-code restrict
11
12   Code bounds  ( start count -- limit start )
13      A pop   A D xchg   D A add   A push   Next   end-code
14 \ : bounds ( start count -- limit start )     over + swap ;
15
```

```
\ 'restart  'cold                              ks 01 sep 88

   Defer 'restart  ' noop Is 'restart

: (restart   ['] (quit Is 'quit  'restart
     [ errorhandler ] Literal @ errorhandler !
     ['] noop Is 'abort  end-trace clearstack
     standardi/o interpret quit ;

   Defer 'cold    ' noop Is 'cold

: (cold      origin up@ $100 cmove   $80 count
     $50 umin >r tib r@ move  r> #tib !  >in off  blk off
     init-vocabularys init-buffers flush 'cold
     Onlyforth page &24 spaces logo count type cr (restart ;
```

1 33

```
0 \ (loop  (+loop                                   ks 27 oct 86
1
2   Code (loop   R ) word inc
3 [[   OS not ?[  4 R D) I mov  ]?  Next   end-code restrict
4
5   Code (+loop   D R ) add   D pop  ]]  end-code restrict
6
7 \\
8
9 ¦ : dodo              rdrop r> 2+ dup >r rot >r swap >r >r ;
10 \ dodo puts "index ¦ limit ¦ adr.of.DO" on return-stack
11
12  : (do  ( limit start -- )  over - dodo ;  restrict
13  : (?do ( limit start -- )  over - ?dup IF dodo THEN
14                          r> dup  @ +  >r drop ; restrict
15
```

1 34

```
0 \ loop indices                                    ks 27 oct 86
1
2   Code I  ( -- n )  D push   R ) D mov   2 R D) D add   Next
3   end-code
4 \ : I     ( -- n )  r>  r> dup r@ + -rot  >r >r ;
5
6   Code J  ( -- n )  D push   6 R D) D mov   8 R D) D add   Next
7   end-code
8
9
10
11
12
13
14
15
```

1 35

```
0 \ branch ?branch                                  ks 27 oct 86
1
2   Code branch
3 [[   I ) I add   Next   end-code restrict
4 \ : branch r> dup @ + >r ;
5
6   Code ?branch  D D or  D pop   0= not ?]
7      I inc   I inc   Next   end-code restrict
8
9
10
11
12
13
14
15
```

107

```
\ (boot                                             ks 11 mär 89

Label #segs  ( -- R: seg )   Assembler
   C: seg ' limit >body #) R mov   R R or  0= not
   ?[  4 # C- mov   R C* shr   R inc   ret  ]?
   $1000 # R mov   ret
end-code

Label (boot   Assembler   cli   cld   A A xor   A D: mov
   #segs # call   C: D mov   D R add   R E: mov
   $200 # C mov   0 # I mov   I W mov   rep movs
     wake # >taskINT #) mov   C: >taskINT 2+ #) mov
   divovl #  >divINT #) mov   C:  >divINT 2+ #) mov   ret
end-code
```

108

```
\ restart                                           ks 09 mär 89

Label warmboot   here >restart 2+ -  >restart ! Assembler
      (boot # call
here   ' (restart >body # I mov
Label bootsystem
   C: A mov   A E: mov   A D: mov   A S: mov
   s0 #) U mov   6 # U add   u' s0 U D) S mov
   D pop   u' r0 U D) R mov   sti   Next
end-code

Code restart   here 2- !   end-code
```

109

```
\ bye                                               ks 11 mär 89

Variable return_code    return_code off

¦ Code (bye   cli   A A xor   A E: mov   #segs # call
   C: D mov   D R add   R D: mov   0 # I mov   I W mov
   $200 # C mov   rep movs   sti      \ restore interrupts
   $4C # A+ mov   C: seg return_code #) A- mov
   $21 int   warmboot # call
end-code

: bye       flush empty page (bye ;
```

1 36 110

```
0 \ resolve loops and branches                    ks 02 okt 87
1
2   : >mark     ( -- addr )         here 0 , ;
3
4   : >resolve  ( addr -- )         here over - swap ! ;
5
6   : <mark     ( -- addr )         here ;
7
8   : <resolve  ( addr -- )         here - , ;
9
10  : ?pairs    ( n1 n2 -- )        - Abort" unstructured" ;
11
12
13
14
15
```

```
\ cold                                            ks 09 mär 89

here  >cold 2+ -   >cold !  Assembler
   (boot # call   C: A mov   A D: mov  A E: mov
   #segs # call   $41 # R add  \ another k for the ints
   $4A # A+ mov   $21 int        \ alloc memory
   CS ?[  $10 # return_code #) byte mov   ' (bye @ # jmp  ]?
here   s0 #) W mov   6 # W add   origin # I mov   $20 # C mov
   rep movs   ' (cold >body # I mov   bootsystem # jmp
end-code

Code cold   here 2- !   end-code
```

1 37 111

```
0 \ Branching                                     ks 17 jul 87
1
2   : IF     compile ?branch >mark  1 ; immediate restrict
3   : THEN   abs 1 ?pairs  >resolve ;   immediate restrict
4   : ELSE   1 ?pairs  compile branch >mark
5            swap >resolve  -1 ;        immediate restrict
6
7   : BEGIN   <mark 2 ;                 immediate restrict
8   : WHILE   2 ?pairs  2 compile ?branch
9            >mark -2 2swap  ;          immediate restrict
10
11 ¦ : (repeat   2 ?pairs  <resolve
12     BEGIN  dup -2 = WHILE  drop >resolve  REPEAT ;
13
14  : REPEAT compile branch   (repeat ; immediate restrict
15  : UNTIL  compile ?branch  (repeat ; immediate restrict
```

```
\ System patchup                                  ks 16 sep 88

1 &35 +thru      \ MS-DOS interface

: forth-83 ;     \ last word in Dictionary

0 ' limit >body !   $DFF6 s0 !    $E77C r0 !
s0 @ s0 2- !   here dp !

Host  tudp @       Target   udp !
Host  tvoc-link @  Target   voc-link !
Host  tnext-link @ Target   next-link !
Host  tfile-link @ Target Forth  file-link !
Host  T move-threads H
save-buffers cr .( unresolved: )  .unresolved
```

1 38 112

```
0 \ Loops                                         ks 27 oct 86
1
2   : DO       compile (do  >mark  3 ; immediate restrict
3   : ?DO      compile (?do >mark  3 ; immediate restrict
4   : LOOP     3 ?pairs  compile (loop
5              compile endloop  >resolve ;  immediate restrict
6   : +LOOP    3 ?pairs  compile (+loop
7              compile endloop  >resolve ;  immediate restrict
8
9   Code LEAVE    6 # R add   -2 R D) I mov
10     I dec   I dec   I ) I add   Next   end-code restrict
11
12 \ : LEAVE     endloop r> 2- dup @ + >r ;        restrict
13 \ Returnstack: ¦ calladr ¦ index ¦ limit ¦ adr of DO ¦
14
15
```

```
\ lc@ lc!  l@ l!  special 8088 operators          ks 27 oct 86

Code lc@  ( seg:addr -- 8b )   D: pop   D W mov
   W ) D- mov   0 # D+ mov   C: A mov   A D: mov   Next
end-code

Code lc!  ( 8b seg:addr -- )   D: pop   A pop   D W mov
   A- W ) mov   C: A mov   A D: mov   D pop   Next   end-code

Code l@  ( seg:addr -- 16b )   D: pop   D W mov
   W ) D mov   C: A mov   A D: mov   Next   end-code

Code l!  ( 16b seg:addr -- )   D: pop   A pop   D W mov
   A W ) mov   C: A mov   A D: mov   D pop   Next   end-code
```

1 39

```
0 \ um*  m*  *                                  ks 29 jul 87
1
2  Code um* ( u1 u2 -- ud3 )
3     A D xchg   C pop   C mul   A push   Next   end-code
4
5  Code m*  ( n1 n2 -- d3 )
6     A D xchg   C pop   C imul   A push   Next   end-code
7 \ : m*  ( n1 n2 -- d )  dup 0< dup >r IF  negate  THEN  swap
8 \    dup 0< IF negate r> not >r THEN  um* r> 0=exit  dnegate ;
9
10  : *      ( n1 n2 - prod )   um* drop ;
11
12  Code 2*  ( u -- 2*u )   D shl   Next   end-code
13 \ : 2*     ( u -- 2*u )   dup + ;
14
15
```

1 40

```
0 \ um/mod  m/mod                               ks 27 oct 86
1
2  Code um/mod  ( ud1 u2 -- urem uquot )
3     D C mov   D pop   A pop   C div   A D xchg   A push   Next
4  end-code
5
6  Code m/mod  ( d1 n2 -- rem quot )   D C mov   D pop
7 Label divide    D+ A+ mov   C+ A+ xor   A pop  0< not
8     ?[  C idiv  [[ swap   A D xchg   A push   Next  ]?
9     C idiv   D D or   dup 0= not ?]  A dec   C D add  ]]
10  end-code
11
12 \ : m/mod ( d n -- mod quot )   dup >r
13 \    abs over 0< IF  under + swap  THEN   um/mod   r@ 0<
14 \    IF  negate over IF  swap r@ + swap 1-  THEN THEN  rdrop ;
15
```

1 41

```
0 \ /mod division trap  2/                      ks 13 sep 88
1
2  Code /mod  ( n1 n2 -- rem quot )
3     D C mov   A pop   cwd   A push   divide ]]  end-code
4 \ : /mod   ( n1 n2 -- rem quot )      over 0< swap m/mod ;
5
6  0 >label >divINT
7
8  Label divovl Assembler
9     4 # S add   popf   1 # D- mov  ;c: Abort" / overflow" ;
10
11  Code 2/  ( n1 -- n/2 )   D sar   Next   end-code
12 \ : 2/  ( n -- n/2 )   2 / ;
13
14
15
```

1 113

```
\ ltype  lmove    special 8088 operators          ks 11 dez 87

  : ltype   ( seg:addr len -- )
    0 ?DO  2dup I + lc@ emit  LOOP  2drop ;

  Code lmove  ( from.seg:addr to.seg:addr quan -- )
    A I xchg   D C mov   W pop   E: pop
    I pop   D: pop   I W cmp  CS
    ?[  rep byte movs
    ][  C dec   C W add   C I add   C inc
        std   rep byte movs   cld
    ]?  A I xchg   C: A mov   A E: mov
    A D: mov   D pop   Next   end-code
```

1 114

```
\  BDOS  keyboard input                          ks 16 sep 88
\ es muss wirklich so kompliziert sein, da sonst kein ^C und ^P

  Variable newkey   newkey off

  Code (key@  ( -- 8b )    D push   newkey #) D mov   D+ D+ or
    0= ?[  $7 # A+ mov   $21 int   A- D- mov  ]?
    0 # D+ mov   D+ newkey 1+ #) mov   Next
  end-code

  Code (key?  ( -- f )    D push   newkey #) D mov   D+ D+ or
    0= ?[  -1 # D- mov   6 # A+ mov   $21 int  0=
          ?[  0 # D+ mov
          ][  -1 # A+ mov   A newkey #) mov   -1 # D+ mov
       ]?  ]?  D+ D- mov   Next
  end-code
```

1 115

```
\ empty-keys  (key                              ks 16 sep 88

  Code empty-keys   $C00 # A mov   $21 int
    0 # newkey 1+ #) byte mov   Next   end-code

  : (key   ( -- 16b )   BEGIN  pause (key?  UNTIL
    (key@ ?dup ?exit  (key? IF  (key@ negate exit  THEN  0 ;
```

1 42

```
0 \ / mod */mod */ u/mod  ud/mod                  ks 27 oct 86
1
2  : /      ( n1 n2 --     quot )      /mod nip ;
3
4  : mod    ( n1 n2 -- rem )           /mod drop ;
5
6  : */mod  ( n1 n2 n3 -- rem quot )   >r m* r> m/mod ;
7
8  : */     ( n1 n2 n3 -- quot )       */mod nip ;
9
10 : u/mod  ( u1 u2 -- urem uquot )    0 swap um/mod ;
11
12 : ud/mod ( ud1 u2 -- urem udquot )
13    >r   0 r@ um/mod   r> swap >r   um/mod r> ;
14
15
```

1 116

```
\\ BIOS  keyboard input                          ks 16 sep 88

Code (key@  ( -- 8b )  D push   A+ A+ xor   $16 int
   A- D- xchg   0 # D+ mov   Next   end-code

Code (key?  ( -- f )   D push   1 # A+ mov   D D xor
   $16 int   0= not ?[  D dec  ]?   Next   end-code

Code empty-keys   $C00 # A mov   $21 int   Next   end-code

: (key  ( -- 8b )   BEGIN  pause (key? UNTIL  (key@ ;

\ mit diesen Keytreibern sind die Funktionstasten nicht
\ mehr durch ANSI.SYS Sequenzen vorbelegt.
```

1 43

```
0 \ cmove cmove> move                            ks 27 oct 86
1
2  Code cmove  ( from to quan -- )   A I xchg   D C mov
3     W pop   I pop   D pop   rep byte movs   A I xchg   Next
4  end-code
5
6  Code cmove>  ( from to quan -- )
7     A I xchg  D C mov  W pop  I pop   D pop
8 Label moveup   C dec   C W add   C I add   C inc
9     std   rep byte movs   A I xchg   cld   Next   end-code
10
11 Code move  ( from to quan -- )
12    A I xchg   D C mov   W pop   I pop   D pop
13 Label domove   I W cmp   moveup CS ?]
14    rep byte movs   A I xchg   Next   end-code
15
```

1 117

```
\ (decode expect                                 ks 16 sep 88

 7 Constant #bel          8 Constant #bs
 9 Constant #tab         $A Constant #lf
$D Constant #cr

: (decode  ( addr pos1 key -- addr pos2 )
   #bs case? IF  dup 0=exit del 1- exit  THEN
   #cr case? IF  dup span ! space   exit  THEN
   >r  2dup +  r@ swap c!  r> emit  1+ ;

: (expect ( addr len1 -- )  span !   0
   BEGIN   dup span @ u< WHILE  key decode  REPEAT  2drop ;

Input: keyboard [ here input ! ]
       (key (key? (decode (expect [ drop
```

1 44

```
0 \ place count                                  ks 27 oct 86
1
2 ¦ Code (place ( addr len to - len to)   A I xchg   D W mov
3     C pop   I pop   C push   W inc   domove ]] end-code
4
5  : place  ( addr len to -)   (place c! ;
6
7  Code count ( addr -- addr+1 len )   D W mov
8     W ) D- mov   0 # D+ mov   W inc   W push   Next   end-code
9
10 \ : move   ( from to quan -- )
11 \    >r  2dup u< IF  r> cmove> exit  THEN  r> cmove ;
12 \ : place  ( addr len to -- ) over >r  rot over 1+  r> move c! ;
13 \ : count ( adr -- adr+1 len ) dup 1+ swap c@ ;
14
15
```

1 118

```
\ MSDOS character output                         ks 29 jun 87

Code charout  ( char -- )   $FF # D- cmp  0= ?[  D- dec  ]?
   6 # A+ mov   $21 int   D pop   ' pause # W mov   W ) jmp
end-code

&80 Constant c/row          &25 Constant c/col

: (emit   ( char -- ) dup bl u< IF  $80 or  THEN  charout ;
: (cr                 #cr charout #lf charout ;
: (del                #bs charout bl charout #bs charout ;
: (at                 2drop ;
: (at?                0 0 ;
: (page               c/col 0 DO  cr  LOOP ;
```

1 45

```
 0 \       fill erase                              ks 27 oct 86
 1
 2  Code fill ( addr quan 8b -- )
 3     D A xchg   C pop   W pop   D pop   rep byte stos   Next
 4  end-code
 5
 6 \ : fill ( addr quan 8b -- )   swap ?dup
 7 \    IF >r over c! dup 1+ r> 1- cmove exit THEN 2drop ;
 8
 9  : erase   ( addr quan --)          0 fill ;
10
11
12
13
14
15
```

1 46

```
 0 \ here allot , c, pad compile                   ks 27 oct 86
 1
 2  Code here ( -- addr )   D push   u' dp U D) D mov   Next
 3  end-code
 4 \ : here    ( -- addr ) dp @ ;
 5
 6  Code allot   ( n -- )   D  u' dp U D) add   D pop   Next
 7  end-code
 8 \ : allot  ( n -- )    dp +! ;
 9
10  : ,      ( 16b -- )  here  ! 2 allot ;
11  : c,     ( 8b -- )   here c! 1 allot ;
12  : pad    ( -- addr ) here $42 + ;
13  : compile           r> dup 2+ >r @ , ; restrict
14
15
```

1 47

```
 0 \ input strings                                 ks 23 dez 87
 1
 2  Variable #tib     #tib off
 3  Variable >tib     here >tib ! $50 allot
 4  Variable >in      >in off
 5  Variable blk      blk off
 6  Variable span     span off
 7
 8  : tib ( -- addr )  >tib @ ;
 9
10  : query     tib $50 expect span @ #tib !  >in off ;
11
12
13
14
15
```

119

```
 \ MSDOS character output                          ks  7 may 85

 : bell   #bel charout ;

 : tipp   ( addr len -- )  bounds ?DO  I c@ emit  LOOP ;

 Output: display [ here output ! ]
         (emit (cr  tipp (del (page (at (at? [ drop
```

120

```
 \ MSDOS printer   I/O Port access                 ks 09 aug 87

 Code lst! ( 8b -- )  $5 # A+ mov   $21 int   D pop   Next
 end-code

 Code pc@    ( port -- 8b )
   D byte in   A- D- mov   D+ D+ xor   Next
 end-code

 Code pc!    ( 8b port -- )
   A pop   D byte out   D pop   Next
 end-code
```

121

```
 \ zero terminated strings                         ks 09 aug 87

 : counted   ( asciz -- addr len )
   dup -1 0 scan drop over - ;

 : >asciz   ( string addr -- asciz )   2dup >r  -
   IF  count r@ place  r@  THEN  0 r> count + c!  1+ ;

 : asciz    ( -- asciz )   name here >asciz ;
```

1 48

```
 0 \ skip scan /string                          ks 22 dez 87
 1
 2  Code skip  ( addr len char -- addr1 len1 )
 3     A D xchg   C pop   C0= not
 4     ?[  W pop   0=rep byte scas   0= not ?[  W dec   C inc  ]?
 5         W push  ]?  C D mov   Next   end-code
 6
 7  Code scan  ( addr0 len0 char -- addr1 len1 )
 8     A D xchg   C pop  C0= not
 9     ?[  W pop  0<>rep byte scas   0= ?[  W dec   C inc  ]?
10         W push  ]?   C D mov   Next   end-code
11
12  Code /string  ( addr0 len0 +n -- addr1 len1 )
13     A pop   C pop   D A sub  CS ?[  A D add   A A xor  ]?
14     C D add   D push   A D xchg   Next   end-code
15
```

1 49

```
 0 \\ scan skip /string                         ks 29 jul 87
 1
 2  : skip ( addr0 len0 char -- addr1 len1 )   >r
 3     BEGIN  dup
 4     WHILE  over c@ r@ = WHILE  1- swap 1+ swap
 5     REPEAT  rdrop ;
 6
 7  : scan ( addr0 len0 char -- addr1 len1 )   >r
 8     BEGIN  dup
 9     WHILE  over c@ r@ - WHILE  1- swap 1+ swap
10     REPEAT  rdrop ;
11
12  : /string ( addr0 len0 +n -- addr1 len1 )
13     over umin rot over + -rot - ;
14
15
```

1 50

```
 0 \ capital                                    ks 19 dez 87
 1
 2  Create (capital  Assembler   $61 # A- cmp  CS not
 3     ?[  $7B # A- cmp  CS not
 4         ?[  $84 # A- cmp  0= ?[  $8E # A- mov  ret  ]?  \ ä
 5             $94 # A- cmp  0= ?[  $99 # A- mov  ret  ]?  \ ö
 6             $81 # A- cmp  0= ?[  $9A # A- mov  ]?  ret  \ ü
 7         ]?  $20 # A- xor
 8     ]?  ret   end-code
 9
10  Code capital ( char -- char' )
11     A D xchg   (capital # call   A D xchg   Next
12  end-code
13
14
15
```

122

```
 \ Disk capacities                              ks 08 aug 88
   Vocabulary Dos   Dos also definitions

   6 Constant #drives

   Create capacities   $480 , $480 , $1831 , $1831 , $180F , 0 ,

 ¦ Code ?capacity ( +n -- cap )  D shl   capacities # W mov
     D W add   W ) D mov   Next   end-code
```

123

```
 \ MS-dos disk handlers direct access           ks 31 jul 87

 ¦ Code block@  ( addr blk drv -- ff )
     D- A- mov   D pop   C pop   R push   U push
     I push   C R mov   2 # C mov   D shl   $25 int
   Label end-r/w   I pop   I pop   U pop   R pop   0 # D mov
     CS ?[  D+ A+ mov   A error# #) mov   D dec  ]?  Next
   end-code

 ¦ Code block!  ( addr blk drv -- ff )  D- A- mov   D pop
     C pop   R push   U push   I push   C R mov   2 # C mov
     D shl   $26 int   end-r/w # jmp
   end-code
```

124

```
 \ MS-dos disk handlers direct access           ks 09 aug 87

 ¦ : ?drive  ( +n -- +n )   dup #drives u< ?exit
     Error" jenseits der Platte" ;

   : /drive ( blk1 -- blk2 drive )  0 swap  #drives 0
     DO  dup I ?capacity under u< IF drop LEAVE THEN
         - swap 1+ swap  LOOP  swap ;

   : blk/drv  ( -- capacity )  drv ?capacity ;

   Forth definitions

   : >drive    ( blk1 +n -- blk2 )   ?drive
     0 swap  drv  2dup u> dup >r  0= IF  swap  THEN
     ?DO  I ?capacity + LOOP  r> IF  negate  THEN - ;
```

1 51

```
 0 \ upper                                          ks 03 aug 87
 1
 2  Code upper   ( addr len -- )
 3     D C mov   W pop   D pop   C0= not
 4     ?[  [[  W ) A- mov   (capital # call
 5             A- W ) mov  W inc  C0= ?]  ]?   Next
 6  end-code
 7
 8 \\ high level, ohne Umlaute
 9
10  : capital ( char -- char')
11     dup  Ascii a   [ Ascii z 1+ ] Literal
12     uwithin not ?exit   [ Ascii a Ascii A - ] Literal - ;
13
14  : upper  ( addr len -- )
15     bounds ?DO  I c@ capital I c!  LOOP ;
```

1 52

```
 0 \ (word                                          ks 28 mai 87
 1
 2 ¦ Code (word  ( char addr0 len0 -- addr1 )   D C mov   W pop
 3     A pop   >in #) D mov   D C sub  >= not
 4     ?[  C push   D W add   0=rep byte scas   W D mov  0= not
 5        ?[  W dec   D dec   C inc
 6           0<>rep byte scas   0= ?[  W dec  ]?
 7        ]?  A pop   C A sub   A >in #) add
 8        W C mov   D C sub  0= not
 9        ?[  D I xchg   u' dp U D) W mov   C- W ) mov
10           W inc   rep byte movs   $20 # W ) byte mov
11           D I mov   u' dp U D) D mov   Next
12 swap ]?  C >in #) add
13        ]?  u' dp U D) W mov   $2000 # W ) mov   W D mov   Next
14  end-code
15
```

1 53

```
 0 \\  (word                                        ks 27 oct 86
 1
 2 ¦ : (word  ( char adr0 len0 -- addr )
 3     rot  >r  over swap   >in @ /string   r@ skip
 4     over swap   r> scan >r   rot over swap - r> 0<> - >in !
 5     over - here  dup >r  place  bl r@ count  + c!  r> ;
 6
 7
 8
 9
10
11
12
13
14
15
```

1 125

```
 0 \ MS-DOS   file access                            ks 18 mär 88
   Dos definitions

 ¦ Variable fcb        fcb off      \ last fcb accessed
 ¦ Variable prevfile                \ previous active file

   &30 Constant fnamelen            \ default length in FCB

   Create filename   &62 allot      \ max 60 + count + null

   Variable attribut   7 attribut !  \ read-only, hidden, system
```

1 126

```
 0 \ MS-DOS   disk errors                           ks 18 mär 88
 1
 ¦ : .error#   ." fehler # " base push decimal error# @ . ;
 ¦ : .ferrors   error# @ &18 case? IF  2  THEN
      1 case? Abort" file exists"
      2 case? Abort" file not found"
      3 case? Abort" path not found"
      4 case? Abort" too many open files"
      5 case? Abort" no access"
      9 case? Abort" beyond end of file"
    &15 case? Abort" illegal drive"
    &16 case? Abort" current directory"
    &17 case? Abort" wrong drive"
    drop ." Disk" .error# abort ;
```

1 127

```
 0 \ MS-DOS   disk errors                           ks 04 okt 87

   : (diskerror   ( *f -- )   ?dup 0=exit
     fcb @ IF  error# !  .ferrors exit  THEN
     input push   output push   standardi/o   1-
     IF  ." Lese"  ELSE  ." Schreib"  THEN
     .error# ."  wiederholen? (j/n)"
     key cr capital Ascii J = not Abort" aborted" ;

   ' (diskerror Is ?diskerror
```

1 54

```
0 \ source word parse name                          ks 03 aug 87
1
2  Variable loadfile     loadfile off
3
4  : source ( -- addr len )   blk @ ?dup
5    IF  loadfile @ (block b/blk  exit  THEN  tib #tib @ exit ;
6
7  : word ( char -- addr )   source (word ;
8
9  : parse ( char -- addr len )   >r  source  >in @ /string
10   over swap   r> scan >r  over - dup  r> 0<>  -  >in +! ;
11
12 : name ( -- string )   bl word dup count upper exit ;
13
14
15
```

128

```
\ ~open  ~creat  ~close                             ks 04 aug 87

Code ~open  ( asciz mode -- handle ff / err# )
   A D xchg   $3D # A+ mov
Label >open   D pop   $21 int   A D xchg
   CS not ?[  D push   0 # D mov  ]?  Next
end-code

Code ~creat  ( asciz attribut -- handle ff / err# )
   D C mov   $3C # A+ mov   >open ]]   end-code

Code ~close   ( handle -- )   D R xchg
   $3E # A+ mov   $21 int   R D xchg   D pop   Next
end-code
```

1 55

```
0 \ state Ascii ," ~lit ("  "                      ks 16 sep 88
1  Variable state   state off
2
3  : Ascii  ( char -- n )   bl word  1+ c@
4    state @ 0=exit   [compile] Literal ; immediate
5
6  : ,"    Ascii " parse  here over 1+ allot place ;
7
8  Code ~lit    ( -- addr )   D push   R ) D mov   D W mov
9    W ) A- mov   0 # A+ mov   A inc   A R ) add   Next
10 end-code restrict
11 \ : ~lit  r> r> under  count + even >r >r ;   restrict
12
13 : ("    ~lit ; restrict
14
15 : "     compile (" ," align ; immediate restrict
```

129

```
\ ~first  ~unlink  ~select  ~disk?                  ks 04 aug 87

Code ~first  ( asciz attr -- err# )
   D C mov   D pop   $4E # A+ mov
[[ $21 int   0 # D mov   CS ?[  A D xchg  ]?   Next
end-code

Code ~unlink  ( asciz -- err# )    $41 # A+ mov  ]] end-code

Code ~select  ( n -- )
   $E # A+ mov   $21 int   D pop   Next   end-code

Code ~disk?  ( -- n )   D push   $19 # A+ mov
   $21 int   A- D- mov   0 # D+ mov   Next
end-code
```

1 56

```
0 \ ." ( .( \ \\ hex decimal                        ks 12 dez 88
1
2  : (."      ~lit count type ; restrict
3  : ."       compile (." ," align ; immediate restrict
4
5  : (        Ascii ) parse 2drop ; immediate
6  : .(       Ascii ) parse type ; immediate
7
8  : \        >in @ negate   c/l mod   >in +! ; immediate
9  : \\       b/blk >in ! ; immediate
10 : have   ( <name> -- f )  name find nip   0<> ; immediate
11 : \needs   have 0=exit  [compile] \  ;
12
13 : hex      $10 base ! ;
14 : decimal  &10 base ! ;
15
```

130

```
\ ~next  ~dir                                       ks 04 aug 87

Code ~next    ( -- err# )   D push   $4F # A+ mov
   $21 int   0 # D mov   CS ?[  A D xchg  ]?   Next
end-code

Code ~dir    ( addr drive -- err# )   I W mov
   I pop   $47 # A+ mov   $21 int   W I mov
   0 # D mov   CS ?[  A D xchg  ]?   Next
end-code
```

1 57

```
0 \ number conversion: digit? accumulate convert    ks 08 okt 87
1
2   : digit? ( char -- digit true/ false )  dup  Ascii 9 >
3     IF  [ Ascii A Ascii 9 - 1- ] Literal -  dup Ascii 9 >  and
4     THEN  Ascii 0 -   dup base @ u<  dup ?exit  nip ;
5
6   : accumulate ( +d0 adr digit -- +d1 adr )   swap >r
7     swap  base @ um* drop   rot  base @ um*  d+   r> ;
8
9   : convert ( +d1 addr0 -- +d2 addr2 )
10    1+  BEGIN  count digit? WHILE  accumulate  REPEAT 1- ;
11
12
13
14
15
```

1 58

```
0 \ number conversion                          ks 29 jun 87
1 ¦ : end?       ( -- flag )             >in @ 0= ;
2
3 ¦ : char       ( addr0 -- addr1 char )   count -1 >in +! ;
4
5 ¦ : previous   ( addr0 -- addr0 char )   1- count ;
6
7 ¦ : punctuation?   ( char -- flag )
8     Ascii , over =   swap Ascii . =  or ;
9 \ : punctuation?  ( char -- f )   ?" .," ;
10
11 ¦ : fixbase?   ( char -- char false / newbase true )  capital
12    Ascii $ case? IF $10 true exit  THEN
13    Ascii H case? IF $10 true exit  THEN
14    Ascii & case? IF &10 true exit  THEN
15    Ascii % case? IF   2 true exit  THEN     false ;
```

1 59

```
0 \ number conversion: dpl ?num ?nonum ?dpl        ks 27 oct 86
1
2   Variable dpl      -1 dpl !
3
4 ¦ : ?num      ( flag -- exit if true )  0=exit
5     rdrop drop r> IF  dnegate  THEN   rot drop
6     dpl @ 1+ ?dup ?exit  drop true ;
7
8 ¦ : ?nonum     ( flag -- exit if true ) 0=exit
9     rdrop 2drop drop rdrop false ;
10
11 ¦ : ?dpl     dpl @  -1 =  ?exit  1 dpl +! ;
12
13
14
15
```

1 131

```
\ MS-DOS file control block                        ks 19 mär 88

¦ : Fcbytes  ( n1 len -- n2 )  Create over c, +
  Does>      ( fcbaddr -- fcbfield )  c@ + ;

\ first field for file-link
2       1 Fcbytes f.no       \ must be first field
        2 Fcbytes f.handle
        2 Fcbytes f.date
        2 Fcbytes f.time
        4 Fcbytes f.size
  fnamelen Fcbytes f.name     Constant b/fcb

b/fcb  Host   ' tb/fcb >body !
       Target Forth also Dos also definitions
```

1 132

```
\ (.file fname  fname!                             ks 10 okt 87

  : fname!   ( string fcb -- )   f.name >r   count
    dup fnamelen < not Abort" file name too long"  r> place ;

¦ : filebuffer?   ( fcb -- fcb bufaddr / fcb ff )
    prev  BEGIN  @ dup WHILE  2dup 2+ @  = UNTIL ;

¦ : flushfile     ( fcb -- )
    BEGIN  filebuffer? ?dup
    WHILE  dup backup emptybuf  REPEAT  drop ;

  : fclose   ( fcb  -- )   ?dup 0=exit
    dup f.handle @ ?dup 0= IF  drop exit  THEN
    over flushfile  ~close  f.handle off ;
```

1 133

```
\ (.file fname  fname!                             ks 18 mär 88

¦ : getsize   ( -- d )     [ $80 &26 + ] Literal 2@ swap ;

  : (fsearch  ( string -- asciz *f )
    filename >asciz dup attribut @ ~first ;

  Defer fsearch   ( string -- asciz *f )

  ' (fsearch Is fsearch

\ graceful behaviour if file does not exist
¦ : ?notfound  ( f* -- )  ?dup 0=exit  last' @  [fcb] =
    IF  hide   file-link @ @ file-link !  prevfile @ setfiles
        last @ 4 - dp !  last off   filename count here place
    THEN  ?diskerror ;
```

1 60

```
0 \ number conversion: number?  number            ks 27 oct 86
1
2  : number?   ( string -- string false / n 0< / d 0> )
3     base push  >in push  dup count >in !  dpl on
4     0 >r ( +sign)   0.0   rot end? ?nonum char
5     Ascii - case?  IF  rdrop true >r end? ?nonum char  THEN
6     fixbase?       IF  base !        end? ?nonum char  THEN
7     BEGIN digit? 0= ?nonum
8           BEGIN  accumulate ?dpl end? ?num char digit?
9        0= UNTIL  previous  punctuation?  0= ?nonum
10          dpl off  end? ?num  char
11    REPEAT ;
12
13 : number ( string -- d )
14    number? ?dup 0= Abort" ?"  0> ?exit  extend ;
15
```

1 61

```
0 \ hide reveal immediate restrict                ks 18 mär 88
1  Variable last     last off
2
3  : last'   ( -- cfa )              last @ name> ;
4
5 ¦ : last?   ( -- false / nfa true)    last @ ?dup ;
6  : hide          last? 0=exit  2- @ current @ ! ;
7  : reveal        last? 0=exit  2-   current @ ! ;
8
9  : Recursive     reveal ; immediate restrict
10
11 ¦ : flag!    ( 8b --)
12     last?  IF  under c@ or over c!  THEN   drop  ;
13
14 : immediate     $40 flag! ;
15 : restrict      $80 flag! ;
```

1 62

```
0 \ clearstack hallot heap heap?                  ks 27 oct 86
1
2  Code clearstack   u' s0 U D) S mov   D pop   Next   end-code
3
4  : hallot  ( quan -- )
5     s0 @  over -  swap    sp@ 2+  dup rot -   dup s0 !
6     2 pick  over -    di  move  clearstack  ei   s0 ! ;
7
8  : heap    ( -- addr )        s0 @ 6 + ;
9  : heap?   ( addr -- flag )   heap up@ uwithin ;
10
11 ¦ : heapmove   ( from -- from )
12     dup  here over -  dup hallot
13     heap swap cmove   heap over - last +!  reveal ;
14
15
```

1 134

```
\ freset fseek                                    ks 19 mär 88

  : freset  ( fcb -- )   ?dup 0=exit
     dup f.handle @ ?dup IF  ~close  THEN   dup >r
     f.name fsearch ?notfound   getsize r@ f.size 2!
     [ $80 &22 + ] Literal @ r@ f.time !
     [ $80 &24 + ] Literal @ r@ f.date !
     2 ~open ?diskerror  r> f.handle ! ;


  Code fseek ( dfaddr fcb -- )
     D W mov   u' f.handle W D) W mov   W W or  0=
     ?[  ;c: dup freset fseek ; Assembler ]?  R W xchg
     C pop   D pop   $4200 # A mov  $21 int   W R mov
     CS not ?[  D pop   Next  ]?  A D xchg  ;c: ?diskerror ;
```

1 135

```
\ lfgets  fgetc  file@                            ks 07 jul 88

  \ Code ~read   ( seg:addr quan handle -- #read )  D W mov
  Assembler  [[   W R xchg   C pop   D pop
      D: pop   $3F # A+ mov   $21 int   C: C mov   C D: mov
      W R mov  A D xchg  CS not ?[  Next  ]?  ;c: ?diskerror ;

  Code lfgets  ( seg:addr quan fcb -- #read )
     D W mov   u' f.handle W D) W mov   ]]  end-code

  true Constant eof

  : fgetc  ( fcb -- 8b / eof )
     >r 0 sp@ ds@ swap 1 r> lfgets ?exit 0= ;

  : file@  ( dfaddr fcb -- 8b / eof )  dup >r fseek r> fgetc ;
```

1 136

```
\ lfputs  fputc  file!                            ks 24 jul 87

¦ Code ~write  ( seg:addr quan handle -- )   D W mov
[[   W R xchg   C pop   D pop
     D: pop   $40 # A+ mov   $21 int   W R mov  A D xchg
     C: W mov   W D: mov  CS ?[  ;c: ?diskerror ; Assembler  ]?
     C D sub  0= ?[  D pop   Next  ]?  ;c: Abort" Disk voll" ;

  Code lfputs  ( seg:addr quan fcb -- )
     D W mov   u' f.handle W D) W mov  ]]  end-code

  : fputc  ( 8b fcb -- )  >r sp@ ds@ swap 1 r> lfputs drop ;

  : file!  ( 8b dfaddr fcb -- )  dup >r fseek r> fputc ;
```

1 63

```
 0 \ Does>  ;                                    ks 18 mär 88
 1
 2 ¦ Create dodo   Assembler
 3     R dec   R dec   I R ) mov      \ push IP
 4     D push   2 W D) D lea          \ load parameter address
 5     W ) I mov   3 # I add   Next   end-code
 6
 7   dodo Host tdodo ! Target       \ target compiler needs to know
 8
 9   : (;code         r> last' ! ;
10
11   : Does>     compile (;code   $E9 c,  ( jmp instruction)
12     dodo here 2+ - , ; immediate restrict
13
14
15
```

1 137

```
 0 \ /block  *block                              ks 02 okt 87
 1
 2   Code /block  ( d -- rest blk )   A D xchg   C pop
 3     C D mov   A shr   D rcr   A shr   D rcr   D+ D- mov
 4     A- D+ xchg   $3FF # C and   C push   Next
 5   end-code
 6 \ : /block  ( d -- rest blk )   b/blk um/mod ;
 7
 8   Code *block  ( blk -- d )  A A xor   D+ D- xchg   D+ A+ xchg
 9     A+ sal   D rcl   A+ sal   D rcl   A push   Next
10   end-code
11 \ : *block  ( blk -- d )   b/blk um* ;
12
13
14
15
```

1 64

```
 0 \ ?head  ¦  alignments                        ks 19 mär 88
 1   Variable ?head     ?head off
 2
 3   : ¦                ?head @  ?exit  ?head on ;
 4
 5   : even   ( addr -- addr1 ) ; immediate
 6   : align  ( -- )            ; immediate
 7   : halign ( -- )            ; immediate
 8 \ machen nichts beim 8088. 8086 koennte etwas schneller werden
 9
10   Variable warning    warning on
11
12 ¦ : ?exists   warning @  0=exit
13     last @ current @ (find nip 0=exit
14     space last @ .name ." exists " ?cr ;
15
```

1 138

```
 0 \ fblock@  fblock!                            ks 19 mär 88
 1   Dos definitions
 2
 3 ¦ : ?beyond   ( blk -- blk )  dup 0< 0=exit  9 ?diskerror ;
 4
 5 ¦ : fblock   ( addr blk fcb -- seg:addr quan fcb )
 6     fcb !  ?beyond dup *block  fcb @  fseek   ds@ -rot
 7     fcb @ f.size 2@ /block rot -  ?beyond
 8     IF  drop b/blk  THEN  fcb @ ;
 9
10   : fblock@  ( addr blk fcb -- )    fblock lfgets drop ;
11
12   : fblock!  ( addr blk fcb -- )    fblock lfputs ;
13
14
15
```

1 65

```
 0 \ Create Variable                             ks 19 mär 88
 1
 2   Defer makeview         ' 0 Is makeview
 3
 4   : Create    align  here  makeview ,  current @ @ ,
 5     name c@ dup 1 $20 uwithin  not Abort" invalid name"
 6     here last !  1+ allot  align   ?exists
 7     ?head @ IF    1 ?head +!   dup ,   \ Pointer to Code
 8                   halign  heapmove   $20 flag!   dup dp !
 9             THEN  drop reveal 0 ,
10   ;Code  ( -- addr )    D push   2 W D) D lea   Next  end-code
11
12   : Variable           Create 0 , ;
13
14
15
```

1 139

```
 0 \ (r/w  flush                                 ks 18 mär 88
 1   Forth definitions
 2
 3   : (r/w   ( addr blk fcb r/wf -- *f )  over fcb !  over
 4     IF  IF  fblock@ false exit  THEN  fblock! false exit
 5     THEN  >r drop  /drive ?drive
 6     r> IF  block@ exit  THEN  block! ;
 7
 8   ' (r/w Is r/w
 9
10 ¦ : setfiles  ( fcb -- )   isfile@ prevfile !
11     dup isfile !   fromfile ! ;
12
13   : direct   0 setfiles ;
14
15
```

1 66

```
0 \ nfa?                                        ks 28 mai 87
1
2  Code nfa?  ( thread cfa -- nfa / false )
3     W pop   R A mov   $1F # C mov
4     [[  W ) W mov   W W or  0= not
5     ?[[  2 W D) R- mov   C R and   3 R W DI) R lea
6          $20 # 2 W D) test  0= not ?[  R ) R mov  ]?
7          D R cmp  0= ?]  2 W D) W lea
8     ]?  W D mov   A R mov   Next   end-code
9
10 \\
11
12  : nfa?   ( thread cfa -- nfa / false )  >r
13     BEGIN  @ dup 0= IF  rdrop exit  THEN
14            dup 2+ name> r@ = UNTIL  2+ rdrop ;
15
```

1 67

```
0 \ >name name> >body .name                     ks 13 aug 87
1
2  : >name   ( acf -- anf / ff )     voc-link
3     BEGIN  @ dup WHILE  2dup 4 - swap nfa?
4            ?dup IF  -rot 2drop exit  THEN  REPEAT nip ;
5
6  : (name>   ( nfa -- cfa )   count  $1F and + even ;
7
8  : name> ( nfa -- cfa )
9     dup (name> swap  c@ $20 and 0=exit  @ ;
10
11 : >body   ( cfa -- pfa )       2+ ;
12 : body>   ( pfa -- cfa )       2- ;
13
14 : .name   ( nfa -- ) ?dup  IF  dup heap?  IF ." ¦ " THEN
15    count $1F and type  ELSE ." ???"  THEN space ;
```

1 68

```
0 \ : ; Constant Variable                       ks 29 oct 86
1
2  : Create:  Create  hide  current @ context !  0 ] ;
3
4  : :        Create:
5  ;Code   R dec   R dec   I R ) mov   2 W D) I lea   Next
6  end-code
7
8  : ;        0 ?pairs   compile unnest   [compile] [   reveal ;
9  immediate restrict
10
11 : Constant ( n -- )   Create ,
12 ;Code      ( -- n )   D push   2 W D) D mov   Next   end-code
13
14
15
```

1 140

```
\ File  >file                                   ks 23 mär 88

: File    Create   file-link @   here file-link ! ,
   here [ b/fcb 2 - ] Literal   dup allot   erase
   file-link @   dup @ f.no c@ 1+   over f.no c!
   last @ count $1F and   rot f.name place
Does> setfiles ;

File kernel.scr    ' kernel.scr @  Constant [fcb]

Dos definitions

: .file   ( fcb -- )
   ?dup IF  body> >name .name exit  THEN  ." direct" ;
```

1 141

```
\ .file  pushfile  close  open                  ks 12 mai 88
Forth  definitions

: file?    isfile@ .file ;

: pushfile    r> isfile push  fromfile push  >r ; restrict

: close    isfile@ fclose ;

: open     isfile@ freset ;

: assign   isfile@ dup fclose   name swap fname!   open ;
```

1 142

```
\      use from loadfrom include                ks 18 mär 88

: use      >in @   name find
   0= IF  swap >in !   File   last' THEN  nip
   dup @ [fcb] =  over ['] direct = or
   0= Abort" not a file"   execute open ;

: from         isfile push   use ;

: loadfrom     ( n -- )   pushfile  use load close ;

: include      1 loadfrom ;
```

1 69

```
 0 \ uallot User Alias Defer                          ks 02 okt 87
 1   : uallot   ( quan -- offset )   even    dup udp @ +
 2     $FF u> Abort" Userarea full"   udp @   swap udp +! ;
 3
 4   : User    Create 2 uallot c,
 5   ;Code   ( -- addr )   D push   2 W D) D- mov
 6                         0 # D+ mov   U D add   Next   end-code
 7
 8   : Alias ( cfa -- )
 9     Create  last @ dup c@ $20 and
10     IF  -2 allot  ELSE  $20 flag!  THEN  (name> ! ;
11
12 ¦ : crash          true Abort" crash" ;
13
14   : Defer     Create ['] crash ,
15   ;Code   2 W D) W mov   W ) jmp   end-code
```

1 70

```
 0 \ vp current context also toss                     ks 02 okt 87
 1
 2   Create vp  $10 allot
 3   Variable current
 4
 5   : context   ( -- adr )          vp dup @ + 2+ ;
 6
 7 ¦ : thru.vocstack ( -- from to )    vp 2+ context ;
 8
 9 \ "Only Forth also Assembler" gives
10 \ vp:  countword = 6 ¦ Root ¦ Forth ¦ Assembler ¦
11
12   : also          vp @ &10 > Error" Vocabulary stack full"
13                   context @  2 vp +!  context ! ;
14
15   : toss          vp @ 0=exit   -2 vp +! ;
```

1 71

```
 0 \ Vocabulary Forth Only Onlyforth definitions      ks 19 jun 88
 1   : Vocabulary  Create  0 , 0 ,  here  voc-link @ ,  voc-link !
 2   Does>   context ! ;
 3 \  ¦ Name ¦ Code ¦ Thread ¦ Coldthread ¦ Voc-link ¦
 4
 5   Vocabulary Forth
 6 Host  h' Transient 8 + @  T h' Forth 8 + H !
 7 Target  Forth also definitions
 8
 9   Vocabulary Root
10
11   : Only     vp off  Root also ;
12
13   : Onlyforth   Only Forth also definitions ;
14
15   : definitions           context @ current ! ;
```

1 143

```
\ drive  drv  capacity   drivenames                  ks 18 mär 88

: drive ( n -- )   isfile@ IF  ~select exit  THEN
  ?drive   offset off 0 ?DO  I ?capacity offset +!  LOOP ;

: drv   ( -- n )
  isfile@ IF  ~disk? exit  THEN  offset @ /drive nip ;

: capacity   ( -- n )   isfile@ ?dup
  IF  dup f.handle @ 0= IF  dup freset  THEN
      f.size 2@ /block swap 0<> - exit  THEN  blk/drv ;

¦ : Drv:   Create c,  Does> c@ drive ;

 0 Drv: A:     1 Drv: B:     2 Drv: C:     3 Drv: D:
 4 Drv: E:     5 Drv: F:     6 Drv: G:     7 Drv: H:
```

1 144

```
\ lfsave  savefile  savesystem                       ks 10 okt 87

: lfsave   ( seg:addr quan string -- )
  filename >asciz 0 ~creat ?diskerror
  dup >r  ~write  r> ~close ;

: savefile ( addr len -- )  ds@ -rot
  name nullstring? Abort" needs name" lfsave ;

: savesystem   save flush   $100 here savefile ;
```

1 145

```
\ viewing                                            ks 19 mär 88
  Dos definitions
¦ $400 Constant viewoffset

  : (makeview   ( -- n )
    blk @ dup 0=exit   loadfile @ ?dup 0=exit   f.no c@ ?dup
    IF  viewoffset * + $8000 or exit  THEN  0= ;
  ' (makeview Is makeview

  : @view  ( acf -- blk fno )   >name 4 - @   dup 0<
    IF  $7FFF and viewoffset u/mod  exit  THEN
    ?dup 0= Error" eingetippt"  0 ;

  : >file   ( fno -- fcb )   dup 0=exit    file-link
    BEGIN  @  dup WHILE  2dup f.no c@ = UNTIL  nip ;
```

1 72

```
 0 \ order vocs words                                  ks 19 jun 88
 1 ¦ : init-vocabularys        voc-link @
 2     BEGIN  dup 2- @ over 4- ! @ ?dup 0= UNTIL ;
 3 ¦ : .voc   ( adr -- )      @ 2- >name .name ;
 4
 5   : order    vp 4+  context over umax
 6     DO  I .voc  -2 +LOOP   2 spaces current .voc ;
 7
 8   : vocs   voc-link
 9     BEGIN  @ ?dup WHILE  dup 6 - >name .name  REPEAT ;
10
11   : words  ( -- )   [compile] Ascii capital >r   context @
12     BEGIN  @ dup  stop? 0=  and
13     WHILE  ?cr dup 2+  r@ bl = over 1+ c@ r@ = or
14       IF  .name space  ELSE  drop  THEN
15     REPEAT drop rdrop ;
```

1 73

```
 0 \ (find  found                                      ks 09 jul 87
 1 ¦ : found ( nfa -- cfa n )   dup c@ >r
 2     (name> r@ $20 and  IF  @       THEN
 3         -1 r@ $80 and  IF  1-      THEN
 4            r> $40 and  IF  negate  THEN ;
 5
 6   Code (find   ( string thread -- string ff / anf tf )
 7     D I xchg   W pop   D push   W ) A- mov   W inc
 8     W D mov   0 # C+ mov   $1F # A+ mov   A+ A- and
 9     [[  I ) I mov   I I or  0= not
10    ?[[  2 I D) C- mov   A+ C- and   A- C- cmp   dup 0= ?]
11         I push   D W mov   3 # I add
12                            0=rep byte cmps   I pop  0= ?]
13         3 # I add   I W mov   -1 # D mov
14     ][  D W mov   0 # D mov  ]?   W dec   I pop   W push   Next
15   end-code
```

1 0

```
 0 \  ####  volksFORTH  ####                           ks 11 mai 88
 1 Entwicklung des volksFORTH-83 von
 2
 3    K. Schleisiek, B. Pennemann, G. Rehfeld, D. Weineck
 4
 5 Zuerst für den 6502 von B.Pennemann und K.Schleisiek
 6 Anpassung für C64 "ultraFORTH"   von G.Rehfeld
 7 Anpassung für 68000   und TOS    von D.Weineck und B.Pennemann
 8 Anpassung für 8080    und CP/M   von U.Hoffmann    jul 86
 9 Anpassung für C16 "ultraFORTH"   von C.Vogt
10 Anpassung für 8088/86 und MS-DOS von K.Schleisiek  dez 87
11
12 Diese Version 3.80 steht auf den aufgeführten Rechnern in
13 identischen Versionen zur Verfügung. Das Fileinterface ist
14 unausgereift und wird in der Version 3.90 entscheidend ver-
15 bessert sein.
```

146

```
 \ forget FCB's                                        ks 23 okt 88
   Forth definitions
 ¦ : 'file  ( -- scr )  r>   scr push   isfile push   >r
     [ Dos ] ' @view >file isfile ! ;

   : view   'file list ;
   : help   'file capacity 2/ + list ;

 ¦ : remove?   ( dic symb addr -- dic symb addr f )
     2 pick over 1+ u< ;

 ¦ : remove-files  ( dic symb -- dic symb )  file-link
     BEGIN  @ ?dup WHILE  remove? IF  dup fclose  THEN  REPEAT
     file-link remove
     isfile@    remove? nip IF  file-link @ isfile !  THEN
     fromfile @ remove? nip 0=exit isfile@ fromfile ! ;
```

147

```
 \ BIOS  keyboard input                                ks 16 sep 88

   Code (key@  ( -- 8b )  D push   A+ A+ xor   $16 int
     0 # D+ mov   A- D- mov   A- A- or
     0= ?[  A+ D- mov   D+ com  ]?   Next   end-code

   : test  BEGIN  (key@ #esc case? ?exit
                  cr dup emit 5 .r  key 5 .r  REPEAT ;
 \\
   Code (key?  ( -- f )   D push   1 # A+ mov   D D xor
     $16 int   0= not ?[  D dec  ]?   Next   end-code

   Code empty-keys   $C00 # A mov   $21 int   Next   end-code

   : (key  ( -- 8b )   BEGIN  pause (key? UNTIL  (key@ ;
```

O

```
 \  ####  volksFORTH  ####                           ks 11 mai 88
 Entwicklung des volksFORTH-83 von

    K. Schleisiek, B. Pennemann, G. Rehfeld, D. Weineck

 Zuerst für den 6502 von B.Pennemann und K.Schleisiek
 Anpassung für C64 "ultraFORTH"   von G.Rehfeld
 Anpassung für 68000   und TOS    von D.Weineck und B.Pennemann
 Anpassung für 8080    und CP/M   von U.Hoffmann    jul 86
 Anpassung für C16 "ultraFORTH"   von C.Vogt
 Anpassung für 8088/86 und MS-DOS von K.Schleisiek  dez 87

 Diese Version 3.80 steht auf den aufgeführten Rechnern in
 identischen Versionen zur Verfügung. Das Fileinterface ist
 unausgereift und wird in der Version 3.90 entscheidend ver-
 bessert sein.
```

1 0

```
 0 \\ Printer Interface                                ks 23 mär 88
 1
 2 Dieses File enthaelt das Printer Interface zwischen volksFORTH
 3 und dem Drucker.
 4
 5 Damit ist es moeglich Source-Texte auf bequeme Art und Weise
 6 in uebersichtlicher Form auszudrucken (6 auf eine Seite).
 7
 8 In Verbindung mit dem Multitasker ist es moeglich, auch Texte im
 9 Hintergrund drucken zu lassen und trotztdem weiterzuarbeiten.
10
11 Diese Druckersteurung wurde von U.Hoffmann für das CP/M
12 volksFORTH geschaffen und von K.Schleisiek verändert.
13
14
15
```

1 1

```
 0 \ Printer Interface M 130i                          ks 08 aug 88
 1   Onlyforth
 2   Vocabulary Printer   Printer definitions also
 3
 4   Variable pcol   pcol off
 5   Variable prow   prow off
 6   Variable prints prints off
 7
 8   2 &10 thru  cr .( Printer Interface für M130i geladen )
 9 \ &11 load    cr .( Spooler geladen )
10
11   : plist  ( scr -- )  prints lock  output push
12      print 10cpi cr list cr 5 lf's  prints unlock ;
13
14   Onlyforth
15
```

1 2

```
 0 \ Printer controls                                  ks 23 mär 88
 1 ¦ : ctrl:  ( 8b --)   Create c,   Does>  ( --)   c@ lst! ;
 2
 3  07   ctrl: bell
 4 $7F ¦ ctrl: ~bs
 5  $D ¦ ctrl: ~cr
 6  $A   ctrl: ~lf
 7  $C   ctrl: ~ff
 8  $F ¦ ctrl: (+17cpi
 9 $1B ¦ ctrl: ESC
10 $12 ¦ ctrl: (-17cpi
11
12 ¦ : esc:  ( 8b --)   Create c,   does>  ( --)    ESC c@ lst! ;
13 ¦ : ESC2  ( 8b0 8b1 --)   #esc lst! lst! ;
14 ¦ : on:  ( 8b --)  Create c, does> ESC c@ lst!  1 lst! ;
15 ¦ : off: ( 8b --)  Create c, does> ESC c@ lst!  0 lst! ;
```

6

```
\ Printer output                                      ks 24 mär 88

  : +emit   dup (emit pemit ;
  : +cr     (cr pcr ;
  : +del    (del pdel ;
  : +page   (page ppage ;
  : +at     2dup (at pat ;

¦ Output: >printer   pemit pcr tipp pdel ppage pat pat? ;
¦ Output: +printer  +emit +cr tipp  +del +page +at (at? ;

Forth definitions

: print   >printer normal ;
: +print  +printer normal ;
```

7

```
\ Variables and Setup                                 ks 09 mai 88

  Printer definitions

  $00 ¦ Constant logo
  ¦ Variable pageno
  ¦ Create scr#s   &14 allot  \ enough room for 6 screens

¦ : header  ( -- )
     normal 4 spaces dark  ." Seite " pageno @ 2 .r
     &13 spaces  ."  volksFORTH83 der FORTH-Gesellschaft eV "
     5 spaces  file?  -dark 1 pageno +! ~lf ;
```

8

```
\ Print 2 screens across on a page                    ks 09 mai 88

¦ : pr  ( scr# -- )   dup  capacity 1- u>
     IF  drop logo  THEN  1 scr#s +!  scr#s dup @ 2* + ! ;

¦ : 2pr  ( scr#1 scr#2 line# -- )
     cr  17cpi dup 2 .r space  c/l * >r
     pad $101 bl fill  swap block  r@ +  pad c/l cmove
     block r> + pad c/l + 1+ c/l cmove
     pad $101 -trailing type ;

¦ : 2scr  ( scr#1 scr#2 -- )  cr cr normal &17 spaces
     wide dark over 4 .r  &28 spaces  dup 4 .r  -wide -dark
     cr  1/s 0 DO  2dup  I 2pr  LOOP  2drop ;

¦ : pr-start  ( --)   scr#s off  1 pageno ! ;
```

1 3

```
 0 \ Printer Escapes                                ks 09 mai 88
 1
 2 Ascii 0   esc: 1/8"       Ascii 1   esc: 1/10"
 3 Ascii 2   esc: 1/6"       Ascii T   esc: suoff
 4 Ascii N   esc: jump       Ascii O   esc: -jump
 5 Ascii G   esc: dark       Ascii H   esc: -dark
 6 Ascii 4   esc: cursive    Ascii 5   esc: -cursive
 7 Ascii M   esc: 12cpi      Ascii P ¦ esc: (-12cpi
 8
 9 : 10cpi   (-12cpi (-17cpi ;
10 : 17cpi   (-12cpi (+17cpi ;
11
12 ' 10cpi   Alias pica
13 ' 12cpi   Alias elite
14
15
```

1 4

```
 0 \ Printer Escapes                                ks 09 mai 88
 1
 2 Ascii W   on:  wide     Ascii W   off: -wide
 3 Ascii -   on:  +under   Ascii -   off: -under
 4 Ascii S   on:  sub      Ascii S   off: super
 5 Ascii p   on:  prop     Ascii p   off: -prop
 6
 7   : lf's   ( n -- )           0 ?DO  ~lf  LOOP ;
 8   : lines  ( #.of.lines -- )  Ascii C ESC2 ;
 9   : "long  ( inches -- )      0 lines lst! ;
10
11   : american   0 Ascii R ESC2 ;
12   : german     2 Ascii R ESC2 ;
13
14   : normal     12cpi  american  suoff 1/6" &12 "long ~cr ;
15
```

1 5

```
 0 \ Printer Output                                 ks 24 mär 88
 1 cr .( verarbeitet Umlaute nicht richtig )
 2   : pemit  ( char -- )  $7F and  1 pcol +!  dup BL u<
 3     IF  $40 or +under lst! -under exit  THEN  lst! ;
 4
 5   : pcr     ~cr ~lf  1 prow +!  pcol off ;
 6
 7   : pdel    ~bs  pcol @ 1- 0 max pcol ! ;
 8
 9   : ppage   ~ff  prow off  pcol off ;
10
11   : pat  ( row col -- )  over  prow @ <  IF  ppage  THEN
12     swap  prow @ -  0 ?DO  pcr  LOOP
13     dup  pcol @ <  IF  ~cr  pcol off  THEN  pcol @ - spaces ;
14
15   : pat?  ( -- row col )  prow @  pcol @ ;
```

9

```
\ Printer 6 screens on a page                      ks 24 mär 88

¦ : pagepr      header  scr#s off  scr#s 2+
     3 0 DO  dup @  over 6 + @  2scr  2+  LOOP  drop  page ;

¦ : shadowpr    header  scr#s off  scr#s 2+
     3 0 DO  dup @  over 2+ @  2scr  4 +  LOOP  drop  page ;

¦ : pr-flush  ( -- f )  \ any screens left over?
     scr#s @  dup 0=exit  0<>
     BEGIN  scr#s @ 5 < WHILE  -1 pr  REPEAT  logo pr ;

¦ Variable shadow

¦ : full?  ( -- f )  scr#s @ 6 = ;
```

10

```
\ Printer 6 screens on a page                      ks 09 mai 88
Forth definitions

: pthru  ( first last -- )  [ Printer ]
   prints lock  output push  print  pr-start  1+ swap
   ?DO  I pr    full? IF  pagepr  THEN  LOOP
   pr-flush IF  pagepr  THEN  prints unlock ;

: document  ( first last -- ) [ Printer ]
   isfile@ IF capacity 2/ shadow ! THEN
   prints lock  output push  print  pr-start  1+ swap
   ?DO  I pr  I shadow @ + pr   full? IF  shadowpr  THEN  LOOP
   pr-flush  IF  shadowpr  THEN  prints unlock ;

: listing    0  capacity 2/ 1- document ;
```

11

```
\ Printerspool                                     ks 30 apr 88

\needs Task \\

¦ Input: noinput    0  false  drop  2drop  ;


noinput  $100 $200 Task spooler  keyboard

: spool   ( from to -- )
   isfile@  spooler 3 pass  isfile !  pthru  stop ;
```

```
1                    O

 0 \\ Startup: Load Standard System                ks 22 dez 87
 1
 2 Dieses File enthaelt Befehle, die aus dem File KERNEL.COM
 3 ein minimales volksFORTH machen.
 4
 5 Dieses System wird unter dem Namen MINIMAL.COM abgelegt.
 6
 7 Es enthält nur einen line orientierten Editor, der dem
 8 Starting Forth Editor entspricht. Mit diesem System kann
 9 EDITOR.SCR und WORK.SCR so geändert werden, daß auch
10 unkompatible Hardware richtig bedient wird.
11
12
13
14
15
```

```
1

\ System LOAD-Screen fuer MS-DOS volksFORTH       ks 09 mai 88
  Onlyforth   warning off

  include extend.scr
  include tools.scr
  include primed.scr

  : initial  primed.scr 0 list restart ;  ' initial Is 'cold

  warning on   clear
  savesystem MINIMAL.COM bell

  .( Neues System ist als MINIMAL.COM abgelegt) cr
```

```
1                    O

 0 \\ Startup: Load Standard System                ks 22 dez 87
 1
 2 Dieses File enthaelt Befehle, die aus dem File KERNEL.COM
 3 ein minimales volksFORTH machen.
 4
 5 Dieses System wird unter dem Namen MINIMAL.COM abgelegt.
 6
 7 Es enthält nur einen line orientierten Editor, der dem
 8 Starting Forth Editor entspricht. Mit diesem System kann
 9 EDITOR.SCR und WORK.SCR so geändert werden, daß auch
10 unkompatible Hardware richtig bedient wird.
11
12
13
14
15
```

```
O

\\ Startup: Load Standard System                ks 22 dez 87

Dieses File enthaelt Befehle, die aus dem File KERNEL.COM
ein minimales volksFORTH machen.

Dieses System wird unter dem Namen MINIMAL.COM abgelegt.

Es enthält nur einen line orientierten Editor, der dem
Starting Forth Editor entspricht. Mit diesem System kann
EDITOR.SCR und WORK.SCR so geändert werden, daß auch
unkompatible Hardware richtig bedient wird.
```

```
1                    O

 0 \\ Startup: Load Standard System                ks 22 dez 87
 1
 2 Dieses File enthaelt Befehle, die aus dem File KERNEL.COM
 3 ein minimales volksFORTH machen.
 4
 5 Dieses System wird unter dem Namen MINIMAL.COM abgelegt.
 6
 7 Es enthält nur einen line orientierten Editor, der dem
 8 Starting Forth Editor entspricht. Mit diesem System kann
 9 EDITOR.SCR und WORK.SCR so geändert werden, daß auch
10 unkompatible Hardware richtig bedient wird.
11
12
13
14
15
```

```
O

\\ Startup: Load Standard System                ks 22 dez 87

Dieses File enthaelt Befehle, die aus dem File KERNEL.COM
ein minimales volksFORTH machen.

Dieses System wird unter dem Namen MINIMAL.COM abgelegt.

Es enthält nur einen line orientierten Editor, der dem
Starting Forth Editor entspricht. Mit diesem System kann
EDITOR.SCR und WORK.SCR so geändert werden, daß auch
unkompatible Hardware richtig bedient wird.
```

1 0

```
 0
 1 This display interface uses BIOS call $10 functions for a fast
 2 display interface. A couple of state variables is contained
 3 in a vector that is task specific such that different tasks
 4 may use different windows. For simplicity windows always
 5 span the whole width of the screen. They can be defined by
 6 top and bottom line. This mechanism is used for a convenient
 7 status display line on the bottom of the screen.
 8
 9
10
11
12
13
14
15
```

1 1

```
 0 \ Multitsking display interface loadscreen        ks  6 sep 86
 1   Onlyforth    \needs Assembler   2 loadfrom asm.scr
 2
 3       User area     area off    \ points at active window
 4   Variable status   \ to switch status on/off
 5 ¦ Variable cursor   \ points at area with active cursor
 6
 7   1 8 +thru   .( Multitasking display aktiv) cr
 8
 9
10
11
12
13
14
15
```

1 2

```
 0 \ Multitsking display interface                   ks  6 sep 86
 1
 2   : Area:  Create  0 , 0 , 7 c, Does> area ! ;
 3 \ ¦ col ¦ row ¦ top ¦ bot ¦ att ¦
 4
 5 Area: terminal    terminal   area @ cursor !
 6
 7   : (area   Create dup c, 1+ Does> c@ area @ + ;
 8
 9 0 ¦ (area ccol    ¦ (area crow    ¦ (area ctop
10   ¦ (area cbot      (area catt    drop
11
12   : window  ( topline botline -- )   cbot c!  ctop c! ;
13
14   : full     0 c/col 2- window ;  full
15
```

1 5

```
\ cur!  curshape  setpage                          ks 28 jun 87

  : cur!   \ set cursor into current task's window
    area @ cursor !  (at? (at ;  cur!

  Code curshape  ( top bot -- )   D C mov   D pop
    D- C+ mov   1 # A+ mov   $10 int   D pop   Next
  end-code

  Code setpage   ( n -- )
    $503 # A mov   D- A- and   $10 int   D pop   Next
  end-code
```

1 6

```
\ Multitask normal invers blankline                ks 01 nov 88
  : normal     7 catt c! ;     : invers   $70 catt c! ;
  : underline  1 catt c! ;     : bright    $F catt c! ;

  Code blankline    D push   R push   U push   $F # A+ mov
    $10 int   u' area U D) W mov   u' catt W D) R- mov
    3 # A+ mov   $10 int   C push   D push
    $E0E # C mov   1 # A+ mov   $10 int   W ) D mov
    2 # A+ mov   $10 int   ' c/row >body #) C mov
    D- C- sub   bl # A- mov   9 # A+ mov
    C- C- or   0= not ?[  $10 int  ]?
    D pop   2 # A+ mov   $10 int  \ set cursor back
    C pop   1 # A+ mov   $10 int  \ cursor visible again
    U pop   R pop   D pop   ' pause #) jmp   end-code

¦ : lineerase  ( line# -- )  0 (at blankline ;
```

1 7

```
\ Multitask (del scroll (cr (page                  ks 04 okt 87

  : (del    (at? ?dup
    IF  1- 2dup (at  bl (emit  (at exit  THEN  drop ;

  Code scroll   D push   R push   U push
    u' area U D) W mov   u' catt W D) R+ mov
    u' ctop W D) D mov   D- C+ mov   0 # C- mov
    ' c/row >body #) D- mov   D- dec   $601 # A mov
    $10 int   U pop   R pop   D pop   Next
  end-code

  : (cr     (at? drop 1+ dup cbot c@ u>
    IF  scroll drop cbot c@  THEN  lineerase ;

  : (page   ctop c@ cbot c@ DO  I lineerase  -1 +LOOP ;
```

1 3

```
 0 \ Multitask (type (emit                          ks 20 dez 87
 1
 2  Code (type   ( addr len -- )   W pop   I push   R push
 3     u' area U D) I mov   U push   D U mov
 4     $F # A+ mov   $10 int   u' catt I D) R- mov
 5     3 # A+ mov   $10 int   C push   D push   $E0E # C mov
 6     1 # A+ mov   $10 int   I ) D mov   1 # C mov
 7     U inc   [[  U dec  0= not ?[[  2 # A+ mov   $10 int
 8        D- inc   ' c/row >body #) D- cmp   0= not
 9     ?[[  W ) A- mov   W inc   9 # A+ mov   $10 int   ]]? ]?
10     D I ) mov   D pop   cursor #) I cmp   0= ?[  I ) D mov  ]?
11     2 # A+ mov   $10 int   C pop   1 # A+ mov   $10 int   U pop
12     R pop   I pop   D pop   ' pause #) jmp   end-code
13
14  : (emit  ( char -- )   sp@ 1 (type drop ;
15
```

1 4

```
 0 \ Multitask (at (at?                             ks 04 aug 87
 1  Code (at    ( row col -- )   A pop   A- D+ mov
 2     u' area U D) W mov   D W ) mov   cursor #) W cmp  0=
 3     ?[  R push   U push   $F # A+ mov   $10 int
 4        2 # A+ mov   $10 int   U pop   R pop
 5     ]?  D pop   Next   end-code
 6
 7  Code (at?   ( -- row col )
 8     D push   u' area U D) W mov   W ) D mov
 9     D+ A- mov   0 # A+ mov   A+ D+ mov   A push   Next
10  end-code
11
12  Code curat? ( -- row col )   D push   R push
13     $F # A+ mov   $10 int   3 # A+ mov   $10 int
14     R pop   0 # A mov   D+ A- xchg   A push   Next
15  end-code
```

1 0

```
 0
 1 This display interface uses BIOS call $10 functions for a fast
 2 display interface. A couple of state variables is contained
 3 in a vector that is task specific such that different tasks
 4 may use different windows. For simplicity windows always
 5 span the whole width of the screen. They can be defined by
 6 top and bottom line. This mechanism is used for a convenient
 7 status display line on the bottom of the screen.
 8
 9
10
11
12
13
14
15
```

8

```
\ Multitask status display                          ks 10 okt 87

  ' (emit ' display   2 + !  ' (cr   ' display   4 + !
  ' (type ' display   6 + !  ' (del  ' display   8 + !
  ' (page ' display &10 + !
  ' (at   ' display &12 + !  ' (at?  ' display &14 + !

  : .base    base @ decimal  dup 2 .r  base ! ;
  : .sp      ( n -- )  ." s" depth   swap 1+ - 2 .r ;
  : (.drv    ( n -- )   Ascii A + emit ." : " ;
  : .dr      ."    " drv (.drv ;
  : .scr     blk @ IF  ."   Blk" blk  ELSE  ."   Scr" scr  THEN
             @ 5 .r ;
  : .space   ."   Dic" s0 @ here $100 + - 6 u.r ;
```

9

```
\ statuszeile                                  ks  ks 04 aug 87

: fstat  ( n -- )   .base .sp
    .space .scr .dr file? 2 spaces order ;

Area: statusline
statusline c/col 1- dup window page invers  terminal

  : (.status   output @ display   area @ statusline
     status @ IF  (at? drop 0 (at 2 fstat  blankline
              ELSE  normal page invers
              THEN  area !  output ! ;
  ' (.status Is .status

  : bye  status off .status bye ;
```

0

This display interface uses BIOS call $10 functions for a fast display interface. A couple of state variables is contained in a vector that is task specific such that different tasks may use different windows. For simplicity windows always span the whole width of the screen. They can be defined by top and bottom line. This mechanism is used for a convenient status display line on the bottom of the screen.

```
1                    0

 0 \\ Printer Interface                              ks 23 mär 88
 1
 2 Dieses File enthaelt das Printer Interface zwischen volksFORTH
 3 und dem Drucker.
 4
 5 Damit ist es moeglich Source-Texte auf bequeme Art und Weise
 6 in uebersichtlicher Form auszudrucken (6 auf eine Seite).
 7
 8 In Verbindung mit dem Multitasker ist es moeglich, auch Texte im
 9 Hintergrund drucken zu lassen und trotztdem weiterzuarbeiten.
10
11 Diese Druckersteurung wurde von U.Hoffmann für das CP/M
12 volksFORTH geschaffen und von K.Schleisiek verändert.
13
14
15
```

```
1                    1

 0 \ Printer Interface NEC 8023 Printer              ks 08 aug 88
 1   Onlyforth
 2   Vocabulary Printer   Printer definitions also
 3
 4   Variable pcol   pcol off
 5   Variable prow   prow off
 6   Variable prints prints off
 7
 8   2 &10 thru  .( Printer Interface für IBM Graphic geladen) cr
 9 \ &11 load    .( Spooler geladen) cr
10
11   : plist  ( scr -- )  prints lock  output push
12      print 10cpi cr list cr 5 lf's  prints unlock ;
13
14   Onlyforth
15
```

```
1                    2

 0 \ Printer controls                                ks 23 mär 88
 1
 2 ¦ : ctrl:   ( char -- )   Create c,   Does> c@ lst! ;
 3
 4
 5   8   ctrl: ~bs
 6  $D   ctrl: ~cr
 7  $A   ctrl: ~lf
 8  $C   ctrl: ~ff
 9 $1B ¦ ctrl: ESC
10 $12   ctrl: 10cpi
11  $F   ctrl: 17cpi
12
13
14
15
```

```
6

\ Printer output                                    ks 24 mär 88

  : +emit   dup (emit pemit ;
  : +cr     (cr pcr ;
  : +del    (del pdel ;
  : +page   (page ppage ;
  : +at     2dup (at pat ;

¦ Output: >printer   pemit pcr tipp pdel ppage pat pat? ;
¦ Output: +printer  +emit +cr tipp  +del +page +at (at? ;

Forth definitions

: print   >printer normal ;
: +print  +printer normal ;
```

```
7

\ Variables and Setup                               ks 09 mai 88

  Printer definitions

  $00 ¦ Constant logo
  ¦ Variable pageno
  ¦ Create scr#s   &14 allot  \ enough room for 6 screens

¦ : header  ( -- )
     normal 4 spaces dark  ." Seite " pageno @ 2 .r
     &13 spaces  ."  volksFORTH83 der FORTH-Gesellschaft eV "
     5 spaces  file?  -dark 1 pageno +! ~lf ;
```

```
8

\ Print 2 screens across on a page                  ks 03 apr 88

¦ : pr  ( scr# -- )   dup  capacity 1- u>
     IF  drop logo  THEN  1 scr#s +!  scr#s dup @ 2* + ! ;

¦ : 2pr  ( scr#1 scr#2 line# -- )
     cr  17cpi dup 2 .r space  c/l * >r
     pad $101 bl fill  swap block  r@ +  pad c/l cmove
     block r> + pad c/l + 1+ c/l cmove
     pad $101 -trailing type ;

¦ : 2scr  ( scr#1 scr#2 -- )  cr cr normal &17 spaces
     wide dark over 4 .r  &18 spaces  dup 4 .r  -wide -dark
     cr  1/s 0 DO  2dup  I 2pr  LOOP  2drop ;

¦ : pr-start  ( --)   scr#s off  1 pageno ! ;
```

1 3

```
 0 \ printer controls                                  ks 24 mär 88
 1
 2 ¦ : #esc:   ( cn..c1 n -- )   Create  dup c, 0 DO c, LOOP
 3   Does> ESC  count bounds DO  I c@ lst!  LOOP ;
 4 $3A 1 #esc: 12cpi
 5
 6 $47 $25 2 #esc: cursive  $48 $25 2 #esc: -cursive
 7 $50 $25 2 #esc: prop     $51 $25 2 #esc: -prop
 8 $33 $49 2 #esc: nlq      $31 $49 2 #esc: standard
 9 $30 $23 2 #esc: fast
10 $31 $57 2 #esc: wide     $30 $57 2 #esc: -wide
11     $47 1 #esc: dark         $48 1 #esc: -dark
12     $32 1 #esc: 6/"          $30 1 #esc: 8/"
13 $31 $2D 2 #esc: +under   $30 $2D 2 #esc: -under
14
15
```

1 4

```
 0 \                                                   ks 24 mär 88
 1
 2
 3
 4   : <rand  ( +n -- )  ESC $58 lst! lst! &300 lst! ;
 5
 6   : lf's   ( +n -- )  0 DO  ~lf  LOOP ;
 7
 8   : normal     standard 12cpi ~cr ;
 9
10
11
12
13
14
15
```

1 5

```
 0 \ Printer output functions                          ks 07 jan 88
 1
 2   : pemit  ( char -- )    1 pcol +!  dup BL u<
 3     IF  $40 or +under lst! -under exit  THEN  lst! ;
 4
 5   : pcr    ~cr ~lf 1 prow +!   pcol off ;
 6
 7   : pdel   ~bs  pcol @ 1- 0 max pcol ! ;
 8
 9   : ppage  ~ff  prow off   pcol off ;
10
11   : pat    ( row col -- )   dup pcol @ -  dup 0< swap
12     abs 0 DO  BL over IF  drop 8  THEN  lst!  LOOP  drop
13     pcol !   prow ! ;
14
15   : pat?   ( -- row col )   prow @   pcol @ ;
```

9

```
 \ Printer 6 screens on a page                         ks 03 apr 88

 ¦ : pagepr      header  scr#s off  scr#s 2+
     3 0 DO  dup @  over 6 + @  2scr  2+  LOOP  drop  page ;

 ¦ : shadowpr    header  scr#s off  scr#s 2+
     3 0 DO  dup @  over 2+ @  2scr  4 +  LOOP  drop  page ;

 ¦ : pr-flush  ( -- f )  \ any screens left over?
     scr#s @  dup 0=exit  0<>
     BEGIN  scr#s @ 5 < WHILE  -1 pr  REPEAT  logo pr ;

 ¦ Variable shadow

 ¦ : full?  ( -- f )  scr#s @ 6 = ;
```

10

```
 \ Printer 6 screens on a page                         ks 09 mai 88
 Forth definitions

 : pthru  ( first last -- )  [ Printer ]
    prints lock  output push  print  pr-start  1+ swap
    ?DO  I pr    full? IF  pagepr  THEN  LOOP
    pr-flush IF  pagepr  THEN  prints unlock ;

 : document  ( first last -- ) [ Printer ]
    isfile@ IF capacity 2/ shadow ! THEN
    prints lock  output push  print  pr-start  1+ swap
    ?DO  I pr  I shadow @ + pr   full? IF  shadowpr  THEN  LOOP
    pr-flush  IF  shadowpr  THEN  prints unlock ;

 : listing    0  capacity 2/ 1- document ;
```

11

```
 \ Printerspool                                        ks 03 apr 88

 \needs Task \\

 ¦ Input: noinput    0  false  drop  2drop  ;

 $100 $200  noinput Task spooler  keyboard

 : spool   ( from to -- )
    isfile@  spooler 3 pass  isfile !  pthru  stop ;
```

1 0

```
\\ Primitivst Editor zur Installation              ks 10 mai 88

Da zur Installationszeit der Full-Screen Editor noch nicht
funktionsfähig ist, müssen die zu ändernden Screens auf eine
andere Weise geändert werden: mit dem Primitivsteditor PRIMED,
der nur ein Benutzerwort enthält:

Benutzung: Mit "nn LIST" Screen nn zum editieren Anwählen,
  dann mit "ll NEW" den Screen ändern. Es können immer nur
  ganze Zeilen neu geschrieben werden. ll gibt an, ab welcher
  Zeilennummer neue Zeilen eingeben werden sollen. Die Eingabe
  einer leeren Zeile (nur RETURN) bewirkt den Abbruch von NEW.
  Nach jeder Eingabe von RETURN wird die eingegebene Zeile in
  den Screen übernommen, und der ganze Screen zur Kontrolle
  noch einmal ausgegeben.
```

1

```
\ primitivst Editor PRIMED                         ks 09 mai 88
 Vocabulary Editor

| : !line ( adr count line# -- )
     scr @ block  swap c/l * +  dup c/l bl fill
     swap cmove update ;

: new ( n -- )
   l/s 1+ swap
   ?DO cr  I .
     pad c/l expect   span @ 0= IF leave THEN
     pad  span @  I !line   cr  scr @ list  LOOP ;

 ' scr | Alias scr'

 .( Primitivsteditor geladen) cr
```

1 0


```
\\ Primitivst Editor zur Installation              ks 10 mai 88

Da zur Installationszeit der Full-Screen Editor noch nicht
funktionsfähig ist, müssen die zu ändernden Screens auf eine
andere Weise geändert werden: mit dem Primitivsteditor PRIMED,
der nur ein Benutzerwort enthält:

Benutzung: Mit "nn LIST" Screen nn zum editieren Anwählen,
  dann mit "ll NEW" den Screen ändern. Es können immer nur
  ganze Zeilen neu geschrieben werden. ll gibt an, ab welcher
  Zeilennummer neue Zeilen eingeben werden sollen. Die Eingabe
  einer leeren Zeile (nur RETURN) bewirkt den Abbruch von NEW.
  Nach jeder Eingabe von RETURN wird die eingegebene Zeile in
  den Screen übernommen, und der ganze Screen zur Kontrolle
  noch einmal ausgegeben.
```


0


```
\\ Primitivst Editor zur Installation              ks 10 mai 88

Da zur Installationszeit der Full-Screen Editor noch nicht
funktionsfähig ist, müssen die zu ändernden Screens auf eine
andere Weise geändert werden: mit dem Primitivsteditor PRIMED,
der nur ein Benutzerwort enthält:

Benutzung: Mit "nn LIST" Screen nn zum editieren Anwählen,
  dann mit "ll NEW" den Screen ändern. Es können immer nur
  ganze Zeilen neu geschrieben werden. ll gibt an, ab welcher
  Zeilennummer neue Zeilen eingeben werden sollen. Die Eingabe
  einer leeren Zeile (nur RETURN) bewirkt den Abbruch von NEW.
  Nach jeder Eingabe von RETURN wird die eingegebene Zeile in
  den Screen übernommen, und der ganze Screen zur Kontrolle
  noch einmal ausgegeben.
```


1 0


```
\\ Primitivst Editor zur Installation              ks 10 mai 88

Da zur Installationszeit der Full-Screen Editor noch nicht
funktionsfähig ist, müssen die zu ändernden Screens auf eine
andere Weise geändert werden: mit dem Primitivsteditor PRIMED,
der nur ein Benutzerwort enthält:

Benutzung: Mit "nn LIST" Screen nn zum editieren Anwählen,
  dann mit "ll NEW" den Screen ändern. Es können immer nur
  ganze Zeilen neu geschrieben werden. ll gibt an, ab welcher
  Zeilennummer neue Zeilen eingeben werden sollen. Die Eingabe
  einer leeren Zeile (nur RETURN) bewirkt den Abbruch von NEW.
  Nach jeder Eingabe von RETURN wird die eingegebene Zeile in
  den Screen übernommen, und der ganze Screen zur Kontrolle
  noch einmal ausgegeben.
```


0


```
\\ Primitivst Editor zur Installation              ks 10 mai 88

Da zur Installationszeit der Full-Screen Editor noch nicht
funktionsfähig ist, müssen die zu ändernden Screens auf eine
andere Weise geändert werden: mit dem Primitivsteditor PRIMED,
der nur ein Benutzerwort enthält:

Benutzung: Mit "nn LIST" Screen nn zum editieren Anwählen,
  dann mit "ll NEW" den Screen ändern. Es können immer nur
  ganze Zeilen neu geschrieben werden. ll gibt an, ab welcher
  Zeilennummer neue Zeilen eingeben werden sollen. Die Eingabe
  einer leeren Zeile (nur RETURN) bewirkt den Abbruch von NEW.
  Nach jeder Eingabe von RETURN wird die eingegebene Zeile in
  den Screen übernommen, und der ganze Screen zur Kontrolle
  noch einmal ausgegeben.
```

1 0

```
 0 \ Extended-Decompiler for VolksForth              ks 11 mai 88
 1
 2 Dieses File enthält einen Decompiler, der bereits kompilierte
 3 Worte wieder in Sourcetextform bringt.
 4 Strukturierte Worte wie IF THEN ELSE, BEGIN WHILE REPEAT UNTIL
 5 und DO LOOP +LOOP werden in einem an AI-grenzenden Vorgang
 6 erkannt und umgeformt.
 7 Ein Decompiler kann aber keine (Stack-) Kommentare wieder
 8 herzaubern, die Benutzung der Screens und dann view, wird
 9 daher stärkstens empfohlen.
10
11 Denn:        Es ist immer noch ein Fehler drin!
12 Und um den zu korrigieren, ist der Sourcetext dem Objektcode
13 doch vorzuziehen.
14
15 Benutzung:   SEE <name>
```

1 1

```
 0 \ Extended-Decompiler for VolksForth LOAD-SCREEN  ks 22 dez 87
 1 Onlyforth Tools also definitions
 2
 3 ¦ : internal    1 ?head ! ;
 4 ¦ : external    ?head off ;
 5
 6 1 &18 +thru
 7
 8 \\
 9 Produces compilable Forth source from normal compiled Forth.
10
11         These source blocks are based on the works of
12
13            Henry Laxen, Mike Perry and Wil Baden
14
15               volksFORTH version: U. Hoffmann
```

1 2

```
 0 \ detecting does>                                 ks 22 dez 87
 1
 2 internal
 3
 4 ' Forth @ 1+ dup @ + 2+ Constant (dodoes>
 5
 6 : does?    ( IP -  f )
 7    dup c@  $E9 ( jmp ) =
 8    swap 1+ dup @ + 2+   (dodoes>  = and ;
 9
10
11
12
13
14
15
```

1 10

```
 \ identify branch destinations.                     ks 22 dez 87
 : ?.then ( ip -- ) thru.branchtable
   ?DO I branch-to @ over =
      IF I branch-from @ over u<
        IF I branch-type @ dup ['] else = swap ['] if = or
          IF -in ." THEN " ind-cr LEAVE THEN THEN THEN
   LOOP ;

 : ?.begin ( ip -- ) thru.branchtable
   ?DO I branch-to @ over =
      IF I branch-from @ over u< not
        IF I branch-type @ dup
           ['] repeat = swap ['] until = or
           IF ind-cr ." BEGIN " +in LEAVE THEN THEN THEN
   LOOP ;
 ( put "BEGIN" and "THEN" where used.)
```

1 11

```
 \ decompile each type of word                       01Jul86

 : .word   ( ip -- ip' ) dup @ >name .name 2+  ;

 : .(word  ( ip -- ip' )     dup @  >name
     ?dup 0= IF ." ??? " ELSE
     count $1f and swap 1+ swap 1-  type space THEN 2+ ;
 : .inline ( val16b -- )
    dup >name ?dup IF ." ['] " .name drop exit THEN . ;

 : .lit    ( ip -- ip' )     2+ dup @ .inline  2+  ?.then ;
 : .clit   ( ip -- ip' )     2+ dup c@ . 1+ ?.then ;
 : .string ( ip -- ip' )
    .(word count 2dup type Ascii " emit space + even ?.then ;

 : .unnest  ( ip -- 0 )      ." ; " 0=  ;
```

1 12

```
 \ decompile each type of word                       01Jul86

 : .default ( ip -- ip' ) dup @ >name ?dup IF
    c@ $40 and IF ." [COMPILE] " THEN THEN   .word ?.then ;

 : .['] ( ip -- ip' )  .(word dup @ 2- >name .name  2+  ?.then ;

 : .compile ( ip -- ip' ) .word  .word  ?.then ;
```

1 3

```
 0 \ indentation.
 1 Variable #spaces   #spaces off
 2
 3 : +in  ( -- ) 3 #spaces +! ;
 4
 5 : -in  ( -- ) -3 #spaces +! ;
 6
 7 : ind-cr ( -- ) ( col #spaces @ = ?exit ) cr  #spaces @ spaces ;
 8
 9 : ?ind-cr ( -- ) col c/l u> IF ind-cr THEN  ;
10
11
12
13
14
15
```

1 4

```
 0 \ case defining words
 1
 2 : Case:  ( -- )
 3    Create:  Does>   swap 2* +  perform ;
 4
 5 : Associative: ( n -- )
 6    Constant   Does>      ( n - index )
 7    dup @ -rot  dup @ 0
 8    DO  2+  2dup @ =
 9        IF  2drop drop  I  0 0  LEAVE  THEN   LOOP 2drop ;
10
11
12
13
14
15
```

1 5

```
 0 \ branching
 1
 2 Variable #branches    Variable #branch
 3
 4 : branch-type ( n -- a ) 6 * pad + ;
 5 : branch-from ( n -- a ) branch-type 2+ ;
 6 : branch-to   ( n -- a ) branch-type 4+ ;
 7
 8 : branched ( adr type -- ) \ Make entry in branch-table.
 9      #branches @ branch-type !   dup #branches @ branch-from !
10      2+ dup @ +  #branches @ branch-to !   1 #branches +! ;
11
12 \\ branch-table: { type0¦from0¦to0 ¦ type1¦from1¦to1 ... }
13
14
15
```

13

```
04Jul86 \ decompiling conditionals                                04Jul86

: .if     ( ip nfa -- ip' )  ind-cr .name +in 4+ ?.then ;
: .repeat ( ip nfa -- ip' )  -in .name ind-cr 4+ ?.then ;
: .else   ( ip nfa -- ip' )  -in ind-cr .name +in 4+ ;
: .do     ( ip nfa -- ip' )  ind-cr .(word +in 2+ ?.then ;
: .loop   ( ip nfa -- ip' )  -in .(word ind-cr 2+ ?.then ;

5 Associative: branch-class
 ' if ,   ' while ,  ' else ,  ' repeat ,  ' until ,
Case: .branch-class
   .if    .else      .else     .repeat     .repeat      ;

: .branch ( ip -- ip' )
    #branch @   branch-type @   1 #branch +!
    dup >name   swap branch-class .branch-class ;
```

14

```
01Jul86 \ decompile Does> ;code                                  04Jul86

: .(;code  ( IP - IP' f)
   2+ dup does?
   IF  ind-cr ." DOES> " 3+  ELSE  ." ;CODE "  0=  THEN ;
```

15

```
04Jul86 \ classify word's output                                 01Jul86

Case: .execution-class
   .clit       .lit          .branch       .branch
   .do         .string       .string       .(;code
   .string     .do           .loop
   .loop       .unnest       .[']          .compile
   .default ;
```

1 6

```
 0 \ branching                                          01Jul86
 1
 2 : branch-back ( adr type --  )
 3   \  : make entry in branch-table & reclassify branch-type.)
 4   over swap branched
 5   2+ dup dup @ + swap 2+ ( loop-start,-end.)
 6   0  #branches @ 1-
 7   ?DO
 8      over I branch-from @  u> IF LEAVE THEN
 9      dup  I branch-to @ = IF ['] while  I branch-type !  THEN
10   -1 +LOOP 2drop ;
11
12
13
14
15
```

1 7

```
 0 \ branching                                          01Jul86
 1 : forward? ( ip -- f )   2+ @ 0> ;
 2
 3 : ?branch+ ( ip -- ip' ) dup 4+ swap dup forward?
 4   IF ['] if branched exit THEN   ['] until branch-back ;
 5
 6 : branch+ ( ip -- ip' ) dup 4+ swap dup forward?
 7   IF ['] else branched exit THEN ['] repeat branch-back ;
 8
 9 : (loop)+ ( ip -- ip' )
10   dup  dup @ ( loop,+loop )  branch-back -1 #branches +! 4+ ;
11
12 : string+ ( ip -- ip' ) 2+ count + even ;
13
14 : (;code+ ( ip -- ip' ) 2+ dup does? not IF 0= exit THEN 3+  ;
15
```

1 8

```
 0 \ classify each word                                 25Aug86
 1 Forth
 2
 3 &15 Associative: execution-class
 4  ] clit        lit           ?branch       branch
 5   (do          (."           (abort"       (;code
 6   ("           (?do          (loop
 7   (+loop       unnest        (is           compile   [
 8
 9 Case:  execution-class+
10   3+           4+            ?branch+      branch+
11   2+           string+       string+       (;code+
12   string+      2+            4+
13   4+           0=            4+            4+        2+  ;
14
15 Tools
```

1 16

```
 0 \ decompile colon-definitions                        04Jul86
 1
 2 : pass2 ( cfa -- ) #branch off >body
 3   BEGIN ?.begin ?ind-cr  dup @ execution-class .execution-class
 4         dup 0= stop? or
 5   UNTIL drop ;
 6
 7 : .pfa ( cfa -- ) #spaces off  +in  dup pass1   pass2 ;
 8
 9 : .immediate  ( cfa - )    >name c@ dup
10   ?ind-cr  40 and IF  ." IMMEDIATE "  THEN
11   ?ind-cr  80 and IF  ." RESTRICT"    THEN ;
12
13 : .:  ( cfa - )  ." : " dup >name .name 3 spaces  .pfa ;
14
15
```

1 17

```
 0 \ display category of word                           01Jul86
 1   external   Defer (see-  internal
 2
 3 : .does>  ( cfa - )     ." DOES> "  @  1+ .pfa ;
 4
 5 : .user-variable  ( cfa - ) ." USER " dup >name dup .name
 6   3 spaces swap execute @ u. .name ." ! " ;
 7
 8 : .defer   ( cfa - )
 9  ." deferred "  dup >name .name  ." Is "  >body @  (see  ;
10
11 : .other   ( cfa - )  dup >name .name
12   dup @ over >body =   IF drop ."  is Code " exit THEN
13   dup @ does? IF .does> exit THEN
14   drop ." is unknown " ;
15
```

1 18

```
 0 \ decompiling variables and constants          ks 22 dez 87
 1
 2 : .constant   ( cfa - )
 3   dup  >body @ u.  ." CONSTANT "  >name .name ;
 4
 5 : .variable   ( cfa - ) ." VARIABLE "
 6   dup >name dup .name 3 spaces swap >body @ u. .name ." ! " ;
 7
 8 5 Associative: definition-class
 9   ' quit @ ,    ' 0 @ ,        ' scr @ ,       ' base @ ,
10   ' 'cold @ ,
11
12 Case: .definition-class
13   .:          .constant      .variable      .user-variable
14   .defer      .other  ;
15
```

1 9

```
 0 \ first pass                                    ks 22 dez 87
 1
 2 : pass1 ( cfa -- ) #branches off >body
 3    BEGIN dup @ execution-class execution-class+
 4       dup 0= stop? or
 5    UNTIL drop ;
 6
 7 : thru.branchtable ( -- limit start ) #branches @ 0 ;
 8
 9
10
11
12
13
14
15
```

1 0

```
 0 \ Extended-Decompiler for VolksForth            ks 11 mai 88
 1
 2 Dieses File enthält einen Decompiler, der bereits kompilierte
 3 Worte wieder in Sourcetextform bringt.
 4 Strukturierte Worte wie IF THEN ELSE, BEGIN WHILE REPEAT UNTIL
 5 und DO LOOP +LOOP werden in einem an AI-grenzenden Vorgang
 6 erkannt und umgeformt.
 7 Ein Decompiler kann aber keine (Stack-) Kommentare wieder
 8 herzaubern, die Benutzung der Screens und dann view, wird
 9 daher stärkstens empfohlen.
10
11 Denn:        Es ist immer noch ein Fehler drin!
12 Und um den zu korrigieren, ist der Sourcetext dem Objektcode
13 doch vorzuziehen.
14
15 Benutzung:   SEE <name>
```

1 0

```
 0 \ Extended-Decompiler for VolksForth            ks 11 mai 88
 1
 2 Dieses File enthält einen Decompiler, der bereits kompilierte
 3 Worte wieder in Sourcetextform bringt.
 4 Strukturierte Worte wie IF THEN ELSE, BEGIN WHILE REPEAT UNTIL
 5 und DO LOOP +LOOP werden in einem an AI-grenzenden Vorgang
 6 erkannt und umgeformt.
 7 Ein Decompiler kann aber keine (Stack-) Kommentare wieder
 8 herzaubern, die Benutzung der Screens und dann view, wird
 9 daher stärkstens empfohlen.
10
11 Denn:        Es ist immer noch ein Fehler drin!
12 Und um den zu korrigieren, ist der Sourcetext dem Objektcode
13 doch vorzuziehen.
14
15 Benutzung:   SEE <name>
```

1 9

```
\ Top level of Decompiler                          ks 20dez87

external

: ((see    ( cfa -)
    #spaces off cr
    dup dup @
    definition-class .definition-class   .immediate  ;

' ((see Is (see

Forth definitions
  : see   ' (see ;
```

0

```
\ Extended-Decompiler for VolksForth            ks 11 mai 88

Dieses File enthält einen Decompiler, der bereits kompilierte
Worte wieder in Sourcetextform bringt.
Strukturierte Worte wie IF THEN ELSE, BEGIN WHILE REPEAT UNTIL
und DO LOOP +LOOP werden in einem an AI-grenzenden Vorgang
erkannt und umgeformt.
Ein Decompiler kann aber keine (Stack-) Kommentare wieder
herzaubern, die Benutzung der Screens und dann view, wird
daher stärkstens empfohlen.

Denn:        Es ist immer noch ein Fehler drin!
Und um den zu korrigieren, ist der Sourcetext dem Objektcode
doch vorzuziehen.

Benutzung:   SEE <name>
```

0

```
\ Extended-Decompiler for VolksForth            ks 11 mai 88

Dieses File enthält einen Decompiler, der bereits kompilierte
Worte wieder in Sourcetextform bringt.
Strukturierte Worte wie IF THEN ELSE, BEGIN WHILE REPEAT UNTIL
und DO LOOP +LOOP werden in einem an AI-grenzenden Vorgang
erkannt und umgeformt.
Ein Decompiler kann aber keine (Stack-) Kommentare wieder
herzaubern, die Benutzung der Screens und dann view, wird
daher stärkstens empfohlen.

Denn:        Es ist immer noch ein Fehler drin!
Und um den zu korrigieren, ist der Sourcetext dem Objektcode
doch vorzuziehen.

Benutzung:   SEE <name>
```

1 0

```
 0 \ Serial interface for IBM-PC using 8250 chip     ks 11 mai 88
 1
 2  INCLUDE SERIAL.SCR    lädt den Code für COM1,
 3  2 LOADFROM SERIAL.SCR für COM2
 4
 5 Empfangene Zeichen werden in einer 128 Byte tiefen Queue
 6 per Interrupt Routine zwischengespeichert.
 7
 8 Die DTR Leitung wird bedient, je nachdem, ob weitere Zeichen
 9 empfangen werden können.
10 Der Sender beachtet CTS, so daß ein vollständiger Hardware-
11 handshake implementiert ist.
12 Xon/Xoff Protokoll mit ^S/^Q ist nicht implementiert.
13
14 Sender:     TX? ( -- f )      TX ( -- char )
15 Empfänger:  RX? ( -- f )      RX ( char -- )
```

1 1

```
 0 \ Driver for IBM-PC Serial card using 8250        ks 11 dez 87
 1  Onlyforth   \needs Assembler  2 loadfrom asm.scr
 2
 3 cr .( COM1: )
 4
 5 ¦ $C 4 * Constant SINT@  \ absolute loc. of serial interrupt
 6
 7  $3F8 >label Portadr
 8
 9 ¦ $10 Constant I_level   \ 8259 priority
10
11  2 7 +thru
12
13
14
15
```

1 2

```
 0 \ Driver for IBM-PC Serial card using 8250        ks 11 dez 87
 1  Onlyforth   \needs Assembler  2 loadfrom asm.scr
 2
 3 cr .( COM2: )
 4
 5 ¦ $B 4 * Constant SINT@  \ absolute loc. of serial interrupt
 6
 7  $2F8 >label Portadr
 8
 9 ¦   8 Constant I_level   \ 8259 priority
10
11  1 6 +thru
12
13
14
15
```

5

```
\ receive queue and interrupt service routine     ks 11 dez 87

  Label S_INT     D push   I push   A push
    Portadr # D mov   D byte in   A- D+ mov
    Queue # I mov   C: seg I ) A mov   A- D- mov   D- inc
    C: seg D- I ) mov   A+ A- add   $7F # A and   A I add
    C: seg D+ 2 I D) mov   $68 # D- cmp  CS not
    ?[  Portadr 4 + # D mov
        D byte in   $1E # A- and   D byte out  ]?  \ -DTR
    $20 # A- mov   I_ctrl #) byte out          \ EOI for 8259
    A pop   I pop   D pop   iret
  end-code
```

6

```
\ rx?  rx                                         ks 30 dez 87

  Code rx?  ( -- f )   D push   D D xor
    Queue #) D- mov   D- D- or  0=
    ?[  [[  D push   Portadr 4 + # D mov   \ +DTR
            D byte in   9 # A- or   D byte out   D pop
swap ]?  Next   end-code

  Code rx   ( -- 8b )  I W mov   Queue # I mov
    D push   D D xor   cli   lods   A- A- or  0= not
    ?[  A+ C- mov   A- dec   A+ inc   $7F # A+ and
        A -2 I D) mov   D- C+ mov   C I add   I ) D- mov
    ]?  sti   W I mov   $18 # A- cmp  CS not ?]   Next
  end-code
```

7

```
\ Serial initialization                           ks 25 apr 86

¦ Code S_init  D push   D: push   A A xor   A D: mov   C: A mov
    SINT@ # W mov   S_INT # W ) mov   A 2 W D) mov   D: pop
    Portadr 3 + # D mov   $80 # A- mov   D byte out \ DLAB = 1
    2 # D sub   baud # A mov   A- A+ xchg   D byte out
    D dec   A- A+ xchg   D byte out                  \ baudrate
    3 # D add   $A07 # A mov   D out     \ 8bit, noP, +RTS +OUT
    2 # D sub   1 # A- mov   D byte out              \ +rxINT
    I_mask #) byte in   I_level Forth not Assembler # A- and
    I_mask #) byte out   D pop   Next
  end-code
```

1 3

```
 0 \ Driver for IBM-PC Serial card using 8250       ks 11 mai 88
 1 \   3 .( 38.4 kbaud )
 2 \  &6 .( 19.2 kbaud )
 3   &12 .( 9.6 kbaud )
 4 \ &24 .( 4.8 kbaud )
 5 \ &96 .( 1200 baud )
 6   >label baud
 7
 8   $20 >label I_ctrl      $21 >label I_mask   \ 8259 addresses
 9
10   Create Queue  0 , $80 allot
11 \  0     1     2                      130  byte address
12 \ ¦ len ¦ out ¦<-- 128 byte Queue -->¦
13 \ len ::= number of characters queued
14 \ out ::= relativ address of next output character
15 \ (len+out)mod(128) ::= relative address of first empty byte
```

1 4

```
 0 \ transmit to 8250                               ks 11 dez 87
 1
 2   Code tx?  ( -- f )   D push   Portadr 5 + # D mov
 3      D in   D D xor   $1020 # A and   $1020 # A cmp
 4      0= ?[  D dec  ]?   Next   end-code
 5
 6   Code tx   ( c -- )   D- A- xchg   Portadr # D mov
 7      D byte out   D pop   Next   end-code
 8
 9   Code -dtr    D push   Portadr 4 + # D mov
10      D byte in   $1E # A- and   D byte out   D pop   Next
11   end-code
12
13   Code +dtr    D push   Portadr 4 + # D mov
14      D byte in   1 # A- or   D byte out   D pop   Next
15   end-code
```

1 0

```
 0 \ Serial interface for IBM-PC using 8250 chip    ks 11 mai 88
 1
 2  INCLUDE SERIAL.SCR    lädt den Code für COM1,
 3  2 LOADFROM SERIAL.SCR für COM2
 4
 5 Empfangene Zeichen werden in einer 128 Byte tiefen Queue
 6 per Interrupt Routine zwischengespeichert.
 7
 8 Die DTR Leitung wird bedient, je nachdem, ob weitere Zeichen
 9 empfangen werden können.
10 Der Sender beachtet CTS, so daß ein vollständiger Hardware-
11 handshake implementiert ist.
12 Xon/Xoff Protokoll mit ^S/^Q ist nicht implementiert.
13
14 Sender:     TX? ( -- f )      TX ( -- char )
15 Empfänger:  RX? ( -- f )      RX ( char -- )
```

8

```
\ init  bye                                        ks 11 dez 87
  \needs init   : init ;

  : init   init   Queue off  S_init ;  init

  : bye    0 [ Portadr 1+ ] Literal  pc!  \ -rxINT
           0 [ Portadr 4 + ] Literal pc!  \ -dtr/-rts/-out2
           I_mask pc@ I_level or I_mask pc! bye ;
```

9

```
\ dumb terminal via 8250                           ks 11 dez 87

  Variable Fkeys   Fkeys on

¦ : ?rx   ( -- )   pause rx? 0=exit rx
     Fkeys @ 0= IF  emit ?cr exit  THEN
     #LF case? IF  cr       exit    THEN
     #CR case? IF  Row 0 at exit    THEN
     #8S case? IF  del exit  THEN   emit ;

¦ : ?tx   ( c -- )   BEGIN ?rx tx? UNTIL tx ;

  : dumb   BEGIN BEGIN  ?rx key?  UNTIL  key
                $1B case? IF -dtr exit THEN ?tx REPEAT ;
```

0


```
\ Serial interface for IBM-PC using 8250 chip      ks 11 mai 88

 INCLUDE SERIAL.SCR    lädt den Code für COM1,
 2 LOADFROM SERIAL.SCR für COM2

Empfangene Zeichen werden in einer 128 Byte tiefen Queue
per Interrupt Routine zwischengespeichert.

Die DTR Leitung wird bedient, je nachdem, ob weitere Zeichen
empfangen werden können.
Der Sender beachtet CTS, so daß ein vollständiger Hardware-
handshake implementiert ist.
Xon/Xoff Protokoll mit ^S/^Q ist nicht implementiert.

Sender:     TX? ( -- f )      TX ( -- char )
Empfänger:  RX? ( -- f )      RX ( char -- )
```

1 0

```
 0 \                                                  ks 16 sep 88
 1 Mit dem Wort STREAM>BLK wird aus einem sequentiellen Eingabe-
 2 file, das mit CR am Zeilenende begrenzt ist, ein Screenfile
 3 zu 64 Zeichen pro Zeile erstellt.
 4
 5 FORTH.TXT sei ein Forthprogramm in einem sequentiellen File.
 6
 7 MAKEFILE FORTH.SCR erzeugt ein leeres File
 8 FROM FORTH.TXT     definiert das Inputfile
 9 STREAM>BLK         überträgt FORTH.TXT ins FORTH.SCR
10
11
12
13
14
15
```

1 1

```
 0 \                                                  ks 06 jul 88
 1   Onlyforth Dos also
 2
 3 ¦ : in   ( -- fcb )  fromfile @ ;
 4 ¦ : out  ( -- fcb )    isfile @ ;
 5
 6 ¦ : padd ( cnt -- )  dup IF  c/l mod ?dup 0=exit  THEN
 7      c/l swap ?DO  BL out fputc  LOOP ;
 8
 9 ¦ : skipctrl  ( -- char )
10      BEGIN  in fgetc dup #cr = ?exit
11             dup 0 BL uwithin 0=exit  drop  REPEAT ;
12
13   2 3 thru
14
15   Onlyforth
```

1 0

```
 0 \                                                  ks 16 sep 88
 1 Mit dem Wort STREAM>BLK wird aus einem sequentiellen Eingabe-
 2 file, das mit CR am Zeilenende begrenzt ist, ein Screenfile
 3 zu 64 Zeichen pro Zeile erstellt.
 4
 5 FORTH.TXT sei ein Forthprogramm in einem sequentiellen File.
 6
 7 MAKEFILE FORTH.SCR erzeugt ein leeres File
 8 FROM FORTH.TXT     definiert das Inputfile
 9 STREAM>BLK         überträgt FORTH.TXT ins FORTH.SCR
10
11
12
13
14
15
```

2

```
\                                                    ks 06 jul 88

¦ : lastline?  ( -- f )   false  0  skipctrl
    BEGIN  -1 case? IF  ?dup IF  padd   THEN  0= exit  THEN
           #cr case? 0= WHILE  out fputc  1+  in fgetc  REPEAT
    padd ;

  : stream>blk   open  out freset
    out f.size 2@ out fseek \ append to end of file
    BEGIN  lastline? stop? or UNTIL  close  out fclose ;
```

3

```
\ absolute blocks in file eintragen                  ks 11 aug 87

¦ : >stream   ( blk -- )
    fromfile @ (block b/blk bounds
    DO  ds@ I C/L -trailing out lfputs
        #cr out fputc  #lf out fputc   C/L +LOOP ;

  : blk>stream  ( from.blk to.blk -- )  emptyfile
    1+ swap DO  I >stream  LOOP  close ;
```

0

```
\                                                    ks 16 sep 88
Mit dem Wort STREAM>BLK wird aus einem sequentiellen Eingabe-
file, das mit CR am Zeilenende begrenzt ist, ein Screenfile
zu 64 Zeichen pro Zeile erstellt.

FORTH.TXT sei ein Forthprogramm in einem sequentiellen File.

MAKEFILE FORTH.SCR erzeugt ein leeres File
FROM FORTH.TXT     definiert das Inputfile
STREAM>BLK         überträgt FORTH.TXT ins FORTH.SCR
```

```
1                    O

 0 \                                                ks 30 apr 88
 1 This file is used for reconfiguring the Forth System
 2
 3 You can only reconfigure a system that does not contain
 4 any additional task.
 5
 6 INCLUDE RECONFIG.SCR will reconfigure and cold-boot the system.
 7
 8 Reconfiguration takes place by overwriting some cold-boot
 9 literals that determine the location of the stacks and the
10 highest address used by the system which happens to be the
11 end of the topmost block-buffer.
12
13 It can be used to tailor the size of an application below 64k
14 and to allocate more stack and/or dictionary space if necessary
15
```

```
1

\ stackdepth returnstackdepth #buffers --          ks 30 apr 88

3 arguments empty

 : reconfigure   ( stack rstack #buffers -- )
    up@ 2+ @ 4+ Abort" no tasks allowed"
    b/buf * >r  2dup + 2+ 0  r> 0 d+
    IF  drop 0  cr ." fewer buffers allocated" bell  THEN
    ['] limit >body !
    over + ['] r0 >body c@ origin + !
       6 - ['] s0 >body c@ origin + !  $80 off  cold ;

 reconfigure
```

```
1                    O

 0 \                                                ks 30 apr 88
 1 This file is used for reconfiguring the Forth System
 2
 3 You can only reconfigure a system that does not contain
 4 any additional task.
 5
 6 INCLUDE RECONFIG.SCR will reconfigure and cold-boot the system.
 7
 8 Reconfiguration takes place by overwriting some cold-boot
 9 literals that determine the location of the stacks and the
10 highest address used by the system which happens to be the
11 end of the topmost block-buffer.
12
13 It can be used to tailor the size of an application below 64k
14 and to allocate more stack and/or dictionary space if necessary
15
```

```
O

\                                                  ks 30 apr 88
This file is used for reconfiguring the Forth System

You can only reconfigure a system that does not contain
any additional task.

INCLUDE RECONFIG.SCR will reconfigure and cold-boot the system.

Reconfiguration takes place by overwriting some cold-boot
literals that determine the location of the stacks and the
highest address used by the system which happens to be the
end of the topmost block-buffer.

It can be used to tailor the size of an application below 64k
and to allocate more stack and/or dictionary space if necessary
```

```
1                    O

 0 \                                                ks 30 apr 88
 1 This file is used for reconfiguring the Forth System
 2
 3 You can only reconfigure a system that does not contain
 4 any additional task.
 5
 6 INCLUDE RECONFIG.SCR will reconfigure and cold-boot the system.
 7
 8 Reconfiguration takes place by overwriting some cold-boot
 9 literals that determine the location of the stacks and the
10 highest address used by the system which happens to be the
11 end of the topmost block-buffer.
12
13 It can be used to tailor the size of an application below 64k
14 and to allocate more stack and/or dictionary space if necessary
15
```

```
O

\                                                  ks 30 apr 88
This file is used for reconfiguring the Forth System

You can only reconfigure a system that does not contain
any additional task.

INCLUDE RECONFIG.SCR will reconfigure and cold-boot the system.

Reconfiguration takes place by overwriting some cold-boot
literals that determine the location of the stacks and the
highest address used by the system which happens to be the
end of the topmost block-buffer.

It can be used to tailor the size of an application below 64k
and to allocate more stack and/or dictionary space if necessary
```

1 0

```
 0 \                                                ks 22 dez 87
 1 The multitasker is a simple yet powerful round robin scheme
 2 with explicit task switching. This has the major advantage
 3 that the system switches tasks only in known states.
 4 Hence the difficulties in synchronizing tasks and locking
 5 critical portions of code are greatly minimized or simply
 6 do not exist at all.
 7
 8
 9
10
11
12
13
14
15
```

1 1

```
 0 \ Multitasker loadscreen                         ks 03 apr 88
 1   Onlyforth   \needs Assembler   2 loadfrom asm.scr
 2
 3   Code stop        $E990 # U ) mov  ' pause @ # jmp   end-code
 4
 5   : singletask   [ ' noop @  ] Literal  ['] pause ! ;
 6   : multitask    [ ' pause @ ] Literal  ['] pause ! ;
 7
 8   1 3 +thru   .( Multitasker geladen) cr
 9
10
11
12
13
14
15
```

1 0

```
 0 \                                                ks 22 dez 87
 1 The multitasker is a simple yet powerful round robin scheme
 2 with explicit task switching. This has the major advantage
 3 that the system switches tasks only in known states.
 4 Hence the difficulties in synchronizing tasks and locking
 5 critical portions of code are greatly minimized or simply
 6 do not exist at all.
 7
 8
 9
10
11
12
13
14
15
```

2

```
\ pass activate                                    ks  1 jun 87

  : pass      ( n0 ... nr-1 Taddr r -- )
BEGIN [ rot ]
    swap $E9CD over !       \ awake Task
    r> -rot                 \ Stack: IP r addr
    8 + >r                  \ s0 of Task
    r@ 2+ @ swap            \ Stack:  IP r0 r
    2+ 2*                   \ bytes on Taskstack incl. r0 & IP
    r@ @ over -             \ new SP
    dup r> 2- !             \ into Ssave
    swap bounds  ?DO  I !  2 +LOOP  ;  restrict

  : activate  ( Taddr -- )  0  \ [ ' pass >body ] Literal >r ;
[ -rot ] REPEAT ; restrict
```

3

```
( Building a Task                            ks  8 may 84 )

: taskerror   ( string -- )   standardi/o  singletask
    ." Task error: " count type multitask stop ;

  : sleep     ( addr -- )    $90 swap c! ;

  : wake      ( addr -- )    $CD swap c! ;

  : rendezvous   ( semaphoraddr -- )
     dup unlock  pause  lock ;
```

O

```
\                                                  ks 22 dez 87
The multitasker is a simple yet powerful round robin scheme
with explicit task switching. This has the major advantage
that the system switches tasks only in known states.
Hence the difficulties in synchronizing tasks and locking
critical portions of code are greatly minimized or simply
do not exist at all.
```

1 0

```
 0 \                                                    ks 22 dez 87
 1
 2 The timer utilizes the memory cell at $46C that is incremented
 3 by an interrupt. A couple of words allow this timer to be
 4 used for time delays.
 5
 6 time-of-day and date are accessed via MS-DOS calls.
 7
 8
 9
10
11
12
13
14
15
```

1 1

```
 0 \ BIMomat BIOS Timer                                 ks 03 apr 88
 1   Onlyforth \needs Assembler    2 loadfrom asm.scr
 2
 3   $46C >label Counter
 4
 5 \ 1193180 / 65536 = 18,206 Hz
 6
 7   1 2 +thru  .( Timer geladen) cr
 8
 9
10
11
12
13
14
15
```

1 0

```
 0 \                                                    ks 22 dez 87
 1
 2 The timer utilizes the memory cell at $46C that is incremented
 3 by an interrupt. A couple of words allow this timer to be
 4 used for time delays.
 5
 6 time-of-day and date are accessed via MS-DOS calls.
 7
 8
 9
10
11
12
13
14
15
```

2

```
\ BIMomat BIOS Timer                                   ks 22 dez 87

  Code ticks ( -- n )   D push   D: C mov   A A xor
     A D: mov   Counter #) D mov   C D: mov   Next   end-code

  : timeout? ( ticks -- ticks f )  pause dup ticks - 0< ;

  : till     ( n -- )        BEGIN  timeout? UNTIL  drop ;

  : time     ( n -- time )   ticks + ;

  : wait     ( n -- )        time till ;

  : seconds  ( sec -- ticks )  &18206 &1000 */ ;

  : minutes  ( min -- ticks )  &1092 * ;
```

3

```
\ MS-DOS time and date                                 ks 22 dez 87

  Code date@  ( -- dd mm yy )
     D push   $2A # A+ mov   $21 int   A A xor   D+ A- xchg
     D push   A push   C D mov   &1900 # D sub   Next
  end-code

  Code time@  ( -- ss mm hh )
     D push   $2C # A+ mov   $21 int   D+ D- mov   0 # D+ mov
     D push   D+ D- mov   C+ D- xchg   C push   Next
  end-code
```

O

```
\                                                      ks 22 dez 87

The timer utilizes the memory cell at $46C that is incremented
by an interrupt. A couple of words allow this timer to be
used for time delays.

time-of-day and date are accessed via MS-DOS calls.
```

1 0

```
 0 \                                                    ks 22 dez 87
 1
 2 Some simple tools for debugging.
 3 A state-of-the-art, interactive single step tracer
 4 and a couple of tools for decompiling and dumping
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

1 1

```
 0 \ Trace Loadscreen                                   ks 03 apr 88
 1   Onlyforth \needs Assembler   2 loadfrom asm.scr
 2
 3   Vocabulary Tools   Tools also definitions
 4
 5   1 9 +thru   Onlyforth   .( Tools geladen) cr
 6
 7
 8
 9
10
11
12
13
14
15
```

1 2

```
 0 \ trace - next                                       ks 11 jun 87
 1
 2 ¦ Variable nest?   nest? off
 3
 4   Label tracenext    0 # nest? #) byte cmp  0=
 5      ?[  $5555 # I cmp  here 2- >label (ip    >=
 6          ?[ [[ swap  lods   A W xchg   W ) jmp  ]?
 7          $5555 # I cmp  here 2- >label ip)    CS ?]
 8      ][  0 # nest? #) byte mov
 9      ]?  $5555 # W mov  here 2- >label >tracing   W ) jmp
10   end-code
11
12 ¦ (ip Constant <ip        ¦ ip) Constant ip>
13
14 ¦ : (debug   ( addr -- )   dup <ip !
15      BEGIN  1+ dup @  ['] unnest = UNTIL  2+ ip> ! ;
```

1 5

```
\ tracer display                                        ks 16 sep 88

¦ : tracing   end-trace  nest? @
     IF  r>  <ip @ >r   ip> @ >r  -nest >r  >r
        1 nest# +!   r@ 2- (debug   nest? off  THEN  r@ 'ip !
     nextstep >r   input @ >r  output @ >r  state @ >r
     blk @ >r  >in @ >r  adr 'quit @ >r  adr parser @ >r
     tib #tib @  rp@ over - under rp! cmove   #tib @ >r
     r0 @ >r   rp@ r0 !   standardi/o
     cr nest# @ spaces   'ip @ dup 5 u.r   @ dup 5 u.r
     2 spaces  >name .name   &30 nest# @ + tab .s
     $20 allot   ['] oneline Is 'quit   quit ;
   ' tracing >tracing !
```

1 6

```
\ test traceability                                     ks 07 dez 87

¦ : traceable  ( cfa -- cfa' )   recursive  dup @
     [ ' :    @ ] Literal case? ?exit
     [ ' key  @ ] Literal case? IF  >body c@ Input @ +
                                    @ traceable exit  THEN
     [ ' type @ ] Literal case? IF  >body c@ Output @ +
                                    @ traceable exit  THEN
     [ ' r/w  @ ] Literal case? IF  >body @ traceable exit  THEN
     c@ $E9 = IF  @ 1+ exit  THEN  \ Does> word
     >name .name ." can't be DEBUGged" quit ;
```

1 7

```
\ user words for tracing                                ks 16 sep 88
¦ : do_debug  ( addr -- )
     traceable (debug  nest? off  nest# off  do-trace ;

   : nest            \ trace next high-level word executed
     'ip @ @ traceable drop  nest? on ;

   : unnest          \ ends tracing of actual word
     <ip on  ip> off ; unnest   \ clears trap range

   : endloop               \ stop tracing loop
     'ip @ <ip ! ;         \ use when at end of loop

   : debug     ' do_debug ;

   : trace     ' dup >r do_debug r> execute end-trace unnest ;
```

1 3

```
 0 \ install Tracer                                  ks 11 jun 87
 1
 2  Label (do-trace   next-link # W mov   D push
 3     $E9 # A- mov   tracenext 1+ # C mov
 4     [[  W ) W mov   W W or  0= not
 5        ?[[  A- -4 W D) mov   C D mov   W D sub
 6             D -3 W D) mov  ]]?  D pop   ret   end-code
 7
 8  Code do-trace   (do-trace # call   Next   end-code
 9
10  ' end-trace Alias end-trace
11
12 ¦ Code (step    (do-trace # call
13     R ) I mov   R inc   R inc   lods   A W xchg   W ) jmp
14
15 ¦ Create: nextstep (step ;
```

1 4

```
 0 \ tracer display                                  ks 20 sep 88
 1
 2 ¦ Variable nest#   nest# off
 3
 4 ¦ Variable 'ip     'ip off
 5
 6 ¦ Create: -nest  r> ip> !   r> <ip !   -1 nest# +! ;
 7
 8 ¦ : oneline   .status space
 9     BEGIN  query  interpret  tib #tib @ + 1- c@ BL =
10     WHILE  prompt &36 tab  REPEAT
11     -$20 allot   r0 @ rp!   r> r0 !   r> dup #tib !
12     rp@ over tib swap cmove   rp@ + rp!
13     r> Is parser   r> adr 'quit !   r> >in !
14     r> blk !   r> state !   r> output !   r> input ! ;
15
```

1 0

```
 0 \                                                 ks 22 dez 87
 1
 2 Some simple tools for debugging.
 3 A state-of-the-art, interactive single step tracer
 4 and a couple of tools for decompiling and dumping
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

8

```
\ tools for decompiling, interactive use          ks 04 jul 87

¦ : ?:  ( addr -- addr )   dup    5 u.r ." :"  ;
¦ : @?  ( addr -- addr )   dup  @ 6 u.r ;
¦ : c?  ( addr -- addr )   dup c@ 3  .r ;
¦ : end                    $28 tab ;

  : s  ( addr1 -- addr2 )
    ?: 3 spaces c? 2 spaces count 2dup type + even end ;
  : n  ( addr1 -- addr2 )
    ?: @? 2 spaces dup @ >name .name 2+ end ;
  : d  ( addr1 n -- addr2 )  2dup swap ?: 3 spaces
    swap 0 DO  c? 1+  LOOP  2 spaces -rot type end ;
  : l  ( addr1 -- addr2 )   ?: 6 spaces @? 2+ end ;
  : c  ( addr1 -- addr2 )   1 d end ;
  : b  ( addr1 -- addr2 )   ?: @? dup @  over + 6 u.r 2+ end ;
```

9

```
\ often  times                                    ks 29 jun 87
  Onlyforth

  : often    stop? ?exit  >in off ;

¦ Variable #times   #times off

  : times   ( n -- )   ?dup
    IF  #times @ 2+  u<  stop? or
        IF  #times off exit  THEN  1 #times +!
    ELSE  stop? ?exit
    THEN  >in off ;
```

0

```
\                                                   ks 22 dez 87

Some simple tools for debugging.
A state-of-the-art, interactive single step tracer
and a couple of tools for decompiling and dumping
```

1 0 2

```
 0 \\ Startup: Load Standard System                ks 22 dez 87
 1
 2 Dieses File enthaelt Befehle, die aus dem File KERNEL.COM
 3 ein vollstaendiges volksFORTH machen.
 4
 5 Dieses System wird unter dem namen WORK.COM abgelegt.
 6
 7 Unter Umständen muß dieses File mit dem im System
 8 MINIMAL.COM enthaltenen Primitiveditor so geändert werden,
 9 daß weniger kompatible Hardware genutzt werden kann.
10
11
12
13
14
15
```

1 1 3

```
 0 \ System LOAD-Screen fuer MS-DOS volksFORTH      ks 30 apr 88
 1   Onlyforth    warning off
 2
 3   include asm.scr
 4   include extend.scr
 5   include multi.vid
 6   include dos.scr
 7   include tasker.scr
 8   include timer.scr
 9   include tools.scr
10   include editor.scr
11   include graphic.prn
12
13   warning on clear status on .status
14   savesystem volks4th.com bell
15   .( Neues System ist als VOLKS4TH.COM abgelegt ) cr
```

1 0 0

```
 0 \\ Startup: Load Standard System                ks 22 dez 87
 1
 2 Dieses File enthaelt Befehle, die aus dem File KERNEL.COM
 3 ein vollstaendiges volksFORTH machen.
 4
 5 Dieses System wird unter dem namen WORK.COM abgelegt.
 6
 7 Unter Umständen muß dieses File mit dem im System
 8 MINIMAL.COM enthaltenen Primitiveditor so geändert werden,
 9 daß weniger kompatible Hardware genutzt werden kann.
10
11
12
13
14
15
```

```
\\ Startup: Load Standard System                ks 22 dez 87

Dieses File enthaelt Befehle, die aus dem File KERNEL.COM
ein vollstaendiges volksFORTH machen.

Dieses System wird unter dem namen WORK.COM abgelegt.

Unter Umständen muß dieses File mit dem im System
MINIMAL.COM enthaltenen Primitiveditor so geändert werden,
daß weniger kompatible Hardware genutzt werden kann.
```