```
\ target words + self compiling kernel

\ This source needs definitely some polishing
\
\ Fri  11-21-1997 fixed z->Z bug in UPPERCASE
\ Wed  01-01-1997 case sensitive per vocabulary
\ 12-19-1996 FROM gone. metacompilation and kernel extension make
\       use of DOS i/o redirection.
\ To metacompile the kernel:
\    TYPE METASTIC.MS' | METASTIC   LOADING ON
\
\ 12-09-96 CASESENSITIVE
\ 07-??-96 swapstore vasm, firstchar wildcard insist on single char,
\          PAD, CREATE doesn't search redef if WARNINGS OFF (speed)

\ 07-10-96 ?sourcefile  recreate prevention
\ 07-07-96 08:34am notebook:   CRESET asm def
\ added some context sensitive definitions :
\  :  ;      ( in TARGET, those compile a high level target definition )
\  ,         ( in TARGET, TCELL determines the number of bytes )

\ ============================================================================
\
\      target cpu                       target cell size
\ #3042 CONSTANT TARGET_CPU             4 CONSTANT TCELL
  #8086 CONSTANT TARGET_CPU             2 CONSTANT TCELL

TCELL CELL/
DUP 0= #IF       ( 32->16 )    : T,  DUP $100 / C, C,  ;   #THEN   1-
DUP 0= #IF       ( nn->nn )    ' ,   ALIAS  T,             #THEN   1-
    0= #IF       ( 16->32 )    : T,  SWAP , ,     ;        #THEN


\ --- target space ---
TARGET_CPU #8086 =    $100 AND   CONSTANT PROGSTART

$8000 VOCABULARY TASM          \ target assembler / compiler words
TASM
$4000 VOCABULARY TARGET-IMAGE  \ target image

: +TARGET ( -- )
  TASM TARGET-IMAGE ;

: TARGET ( -- )
  +TARGET SEARCH OFF ;

TARGET
 CONTEXT @
 HERE

TASM CONSTANT T0       \ target origin
: ?TARGET  ( -- f )
  LITERAL CONTEXT @ =  ;
```

```
DOES:
  ?TARGET IF            ( compiling to target )
    T,
  ELSE                  ( host )
    ,
  THEN  ;
ALIAS ,


: >TARGET ( addr -- )
  T0 +  ;

: THERE  ( -- addr )
  CONTEXT @ TARGET
  HERE T0 - SWAP
  CONTEXT !  ;
' THERE ALIAS $

: SAVE-TARGET  ( -- )
  PROGSTART >TARGET
  TARGET FIRST @ OVER -  BSAVE
  "can't save target image" ?ERROR
  BYE  ;


\ --- target assembler words ---
\ MULTI-PLATFORM SUPPORT - kind of conditional compilation:

: OPCODE_COMPILER  ( cpu -- )  ( stream: commands )
  DUP TARGET_CPU <> AND
  IF
    BEGIN
      NEXTWORD ";" MATCH$
    UNTIL
  ELSE
    [COMPILE] :
  THEN   ;

DOES:    AR> @ OPCODE_COMPILER ;
: SPECIFIC_CODE ( cpu -- )
  BUILD , ;

#8086 SPECIFIC_CODE 8086:
#3042 SPECIFIC_CODE 3042:
0     SPECIFIC_CODE ANY:

\ register use:       8086    3042
\ sp  (data stack)     SP      ER7
\ rp  (return stack)   BP      ER6
\ ip  (instr pointer)  SI      ER5
\ wp  (word pointer)   AX      ER4
\ i (inner loop index) CX      ER3
\ tos (stack cache)    BX      ER0
\ temp1 (temporary)    DI      ER4
\ temp2 (temporary)    DX      -
```

```
\ unused                --      ER1,ER2


8086: rp+               $45 C,  $45 C,            ; \ inc bp inc bp
8086: -rp               $4D C,  $4D C,            ; \ dec bp dec bp

8086: mov(i,tos)        $8B C, $D9 C,             ; \ mov bx,cx
8086: mov(tos,i)        $8B C, $CB C,             ; \ mov cx,bx
8086: mov(tos,temp1)     $89 C, $DF C,             ; \ mov di,bx
8086: mov(temp1,tos)     $89 C, $FB C,             ; \ mov bx,di
8086: mov(tos,wp)       $8B C, $C3 C,             ; \ mov ax,bx
8086: mov(wp,tos)       $8B C, $D8 C,             ; \ mov bx,ax
8086: mov(sp,temp1)      $8B C, $FC C,             ; \ mov di,sp


8086: mov([rp],tos)     $8B C, $5E C, 0 C,        ; \ mov bx,[bp]
8086: mov(tos,[rp])     $89 C, $5E C, 0 C,        ; \ mov [bp],bx

8086: mov([sp],temp1)    $5F C, $57 C,             ; \ pop di  push di
8086: mov([temp1],tos)   $8B C, $1D C,             ; \ mov bx,[di]
8086: mov([tos],tos)    $8B C, $1F C,             ; \ mov bx,[bx]
8086: mov([rp],ip)      $8B C, $76 C, $00 C,      ; \ mov si,[bp]


8086: mov(i,-[rp])      -rp $89 C, $4E C, 00 C,   ; \ mov -[bp],cx
8086: mov([rp]+,i)      $8B C, $4E C, $00 C, rp+  ; \ mov cx,[bp]+

8086: mov(ip,-[rp])     -rp $89 C, $76 C, 0 C,    ; \ mov -[bp],si
8086: mov([rp]+,ip)     mov([rp],ip)   rp+        ; \ mov si,[bp]+

8086: mov(tos,-[rp])    -rp  mov(tos,[rp])        ;
8086: mov([rp]+,tos)    mov([rp],tos)  rp+        ;



8086: mov(ip,-[sp])     $56 C,                    ; \ push si
8086: mov([sp]+,ip)     $5E C,                    ; \ pop si

8086: mov(temp1,-[sp]) $57 C,                     ; \ push di
8086: mov([sp]+,temp1) $5F C,                     ; \ pop di

8086: mov(temp2,-[sp]) $52 C,                     ; \ push dx
8086: mov([sp]+,temp2) $5A C,                     ; \ pop dx

8086: mov(tos,-[sp])    $53 C,                    ; \ push bx
8086: mov([sp]+,tos)    $5B C,                    ; \ pop bx

8086: mov(wp,-[sp])     $50 C,                    ; \ push ax
8086: mov([sp]+,wp)     $58 C,                    ; \ pop ax


8086: mov([ip]+,wp)     $AD C,                    ; \ lodsw
8086: mov([sp]+,[tos]) $8F C, $07 C,              ; \ pop [bx]
```

```
\ monadic tos
8086: inc(tos)         $43 C,                    ; \ inc bx
8086: dec(tos)         $4B C,                    ; \ dec bx
8086: neg(tos)         $F7 C, $DB C,             ; \ neg bx
8086: not(tos)         $F7 C, $D3 C,             ; \ not bx
8086: test(tos)        $85 C, $DB C,             ; \ test bx
8086: clr(tos)         $33 C, $DB C,             ; \ xor bx,bx
8086: asr(tos)         $D1 C, $FB C,             ; \ sar bx,1
8086: lsr(tos)         $D1 C, $EB C,             ; \ shr bx,1
8086: lsl(tos)         $D1 C, $E3 C,             ; \ shl bx,1

\ diadic temp1,tos
8086: add(temp1,tos)    $03 C, $DF C,             ; \ add bx,di
8086: sub(temp1,tos)    $2B C, $DF C,             ; \ sub bx,di
8086: and(temp1,tos)    $23 C, $DF C,             ; \ and bx,di
8086: or(temp1,tos)     $0B C, $DF C,             ; \ or bx,di
8086: xor(temp1,tos)    $33 C, $DF C,             ; \ xor bx,di

\ diadic temp1,[tos]
8086: swap(temp1,[tos]) $87 C, $3F C,             ; \ xchg [bx],di

\ loops
8086: ?bra(-i)  ( dest -- )  $E2 C, C,            ; \ loop dest
8086: mov(#,i)   ( n -- )    $B9 C, ,             ; \ mov cx,n

\ branches
8086: add(tos,ip)      $03 C, $F3 C,              ; \ add si,bx
8086: add([ip],ip)     $03 C, $34 C,              ; \ add si,[si]
8086: bra(wp)          $FF C, $E0 C,              ; \ jmp ax


\ --- target compiler words ---
: org   ( addr -- )
  CONTEXT @ TARGET
  SWAP T0 + -ALLOT
  CONTEXT !  ;

DOES: AR> @ , ;
: comp: ( -- )   ( stream )  ( run time: -- )
  CONTEXT @
  THERE TASM BUILD ,
  CONTEXT !  ;

: l:   ( -- )   ( stream )  ( run time: -- addr )
  CONTEXT @
  THERE TASM CONSTANT
  CONTEXT !  ;

: DATA:  ( -- )
  0 C,
  LINECOUNTER @ ,
  CREATE THERE LATEST !  ;
```

```
: EXEC:  ( -- )
  DATA: EXECUTABLE  ;


: offs     ( addr -- off )
  THERE -  ;

: code     ( -- )   ( stream )
  comp:  ;

: endcode  ( -- )
  mov([ip]+,wp)
  bra(wp)  ;

8086: relative ( addr -- off )
  THERE CELL+ -  ;

8086: relative8 ( addr -- off )
  relative 1+ ;

8086: beq
  $74 C,  relative8 C, ;       \ JE offs8

8086: bne
  $75 C,  relative8 C, ;       \ JNE offs8

8086: bra
  $E9 C,  relative , ;        \ JMP offs16

8086: call
  $E8 C,  relative , ;        \ CALL offs16

#32 CONSTANT MAXSOURCEFILES

\ ------------------------------------------------------------------------
\ ---                      target program                              ---
\ ------------------------------------------------------------------------

TARGET WARNINGS OFF        \ many names already defined
PROGSTART org
l: metastic   0 bra        \ patched to addr of main program when defined
l: byevector  $ TCELL + ,

code return_to_dos ( returncode -- )
  $B4 C, $4C C,            \ mov ah,04c
  $8A C, $C3 C,            \ mov al,bl
  $CD C, $21 C,            \ int $21




\ --- virtual machine & defining words ---
DATA: DOCONST
l: doconst
  mov([sp]+,temp1)
```

```
  mov(tos,-[sp])
  mov([temp1],tos)
endcode

DATA: DOVAR
l: dovar
  mov([sp]+,temp1)
  mov(tos,-[sp])
  mov(temp1,tos)
endcode

DATA: DOCOL
l: docolon
  mov(ip,-[rp])
  mov([sp]+,ip)
endcode


EXEC: EXIT
code exit  ( -- )
l: dosemi
  mov([rp]+,ip)
endcode



TASM
DOES:
  ?TARGET IF            ( compiling to target )
    comp:
    docolon call
  ELSE                  ( host )
    [COMPILE] :
  THEN  ;
ALIAS : IMMEDIATE


DOES:
  ?TARGET IF          ( compiling to target )
    dosemi ,
  ELSE
    [COMPILE] ;
  THEN  ;
ALIAS ; IMMEDIATE


: constant  ( n -- ) ( stream )
  comp: doconst call , ;

: create    ( -- )   ( stream )
  comp: dovar call ;

: variable  ( -- )   ( stream )
  create 0 , ;
```

```
TARGET

EXEC: SS0       create ss0      THERE     0 ,
EXEC: DSEGSIZE  create dsegsize THERE $0FFF ,      \ $1000 paragraphs = 64k
EXEC: RP0       create rp0      THERE     0 ,
EXEC: RSSIZE    create rssize   THERE  $100 ,      \ **stack
EXEC: SP0       create sp0      THERE     0 ,
EXEC: DSSIZE    create dssize   THERE  $200 ,      \ **stack


code initstacks  ( ? -- )                       \ **stack
  $A1 C, ,                    \ mov ax,[dssize]
  $A3 C, ,                    \ mov [sp0],ax
  $8B C, $1E C, ,             \ mov bx,[rssize]
  $01 C, $C3 C, $89 C,        \ add bx,ax
  $1E C, ,                    \ mov [rp0],bx
  $8C C, $CA C, $03 C,        \ mov dx,cs
  $16 C, ,                    \ add dx,[dsegsize]
  $89 C, $16 C, ,             \ mov [ss0],dx
  $8E C, $D2 C,               \ mov ss,dx
  $8B C, $EB C,               \ mov bp,bx
  $89 C, $C4 C,               \ mov sp,ax
endcode



EXEC: DEBUG
code debug  ( -- )
 $CC C,
endcode

\ --- platform transparent target primitives ---
EXEC: DUP
code dup  ( n -- n n )
  mov(tos,-[sp])
endcode

EXEC: DROP
code drop  ( n -- )
  mov([sp]+,tos)
endcode

EXEC: NIP
code nip  ( m n -- n )
  mov([sp]+,temp1)
endcode

EXEC: 1+
code oneplus  ( n -- n+1 )
  inc(tos)
endcode

EXEC: 2+
```

```
code twoplus  ( n -- n+2 )
  inc(tos)
  inc(tos)
endcode

EXEC: 1-
code oneminus  ( n -- n-1 )
  dec(tos)
endcode

EXEC: 2-
code twominus  ( n -- n-2 )
  dec(tos)
  dec(tos)
endcode



EXEC: 2DROP
code 2drop  ( m n -- )
  mov([sp]+,tos)
  mov([sp]+,tos)
endcode


EXEC: 3DROP
code 3drop  ( m n -- )
  mov([sp]+,tos)
  mov([sp]+,tos)
  mov([sp]+,tos)
endcode



EXEC: >R      THERE
code >r   ( n -- )
  mov(tos,-[rp])
  mov([sp]+,tos)
endcode       ALIAS >AR

EXEC: R@      THERE
code r@   ( -- n )
  mov(tos,-[sp])
  mov([rp],tos)
endcode       ALIAS AR@

EXEC: R>  THERE
code r>   ( -- n )
  mov(tos,-[sp])
  mov([rp]+,tos)
endcode   ALIAS AR>

EXEC: I
code i    ( -- n )
  mov(tos,-[sp])
```

```
  mov(i,tos)
endcode

EXEC: EXECUTE
code execute  ( ? addr -- ? )
  mov(tos,wp)
  mov([sp]+,tos)
  bra(wp)
endcode

EXEC: SWAP
code swap  ( m n -- n m )
  mov([sp]+,temp1)
  mov(tos,-[sp])
  mov(temp1,tos)
endcode

EXEC: TUCK
code tuck  ( m n -- n m n )
  mov([sp]+,temp1)
  mov(tos,-[sp])
  mov(temp1,-[sp])
endcode

EXEC: OVER
code over  ( m n -- m n m )
  mov([sp],temp1)
  mov(tos,-[sp])
  mov(temp1,tos)
endcode

EXEC: 2DUP
code 2dup  ( m n -- m n m n )
  mov([sp],temp1)
  mov(tos,-[sp])
  mov(temp1,-[sp])
l: nodup   ( borrowing endcode for ?dup )
endcode

EXEC: ?DUP  ( 0 -- 0  ||  n -- n n )
code ?dup
  test(tos)
  nodup beq
  mov(tos,-[sp])
endcode


EXEC: 3DUP
code 3dup  ( n1 n2 n3 -- n1 n2 n3 n1 n2 n3 )
  mov([sp]+,wp)
  mov([sp],temp1)
  mov(wp,-[sp])
  mov(tos,-[sp])
  mov(temp1,-[sp])
  mov(wp,-[sp])
```

```
endcode


EXEC: FOR
code for  ( n -- )
  mov(i,-[rp])
  mov(ip,-[rp])
  mov(tos,i)
  mov([sp]+,tos)
endcode


\ --- target primitives ---
l: moreloop
  mov([rp],ip)
endcode

EXEC: NEXT
code next  ( -- )
  moreloop relative ?bra(-i)
  rp+                           \ drop loop start address
  mov([rp]+,i)                  \ restore prev i
endcode

EXEC: LEAVE
code leave  ( -- )
   1 mov(#,i)
endcode

EXEC: NOOP
code noop  ( -- )
endcode

EXEC: NEGATE
code negate  ( n -- -n )
  neg(tos)
endcode

EXEC: @
code fetch  ( addr -- n )
  mov([tos],tos)
endcode

EXEC: !
code store  ( n addr -- )
  mov([sp]+,[tos])
  mov([sp]+,tos)
endcode

EXEC: +
code plus  ( m n -- n )
  mov([sp]+,temp1)
  add(temp1,tos)
endcode
```

```
EXEC: -
code minus   ( m n -- n )
  mov(tos,temp1)
  mov([sp]+,tos)
  sub(temp1,tos)
endcode

EXEC: 2/
code two/  ( n -- n/2 )
  asr(tos)
endcode

EXEC: 2*
code two*  ( n -- 2n )
  lsl(tos)
endcode

EXEC: AND
code and  ( m n -- n )
  mov([sp]+,temp1)
  and(temp1,tos)
endcode

EXEC: OR
code or   ( m n -- n )
  mov([sp]+,temp1)
  or(temp1,tos)
endcode

EXEC: XOR
code xor  ( m n -- n )
  mov([sp]+,temp1)
  xor(temp1,tos)
endcode


EXEC: NOT
code not  ( n -- n )
  not(tos)
endcode


EXEC: <>
code notequal  ( m n -- f )                    \ TRY:
  mov([sp]+,temp1)       \ POP DI              \   POP DI
  sub(temp1,tos)         \ SUB BX,DI           \   SUB DI,BX
   $33 C,  $C0 C,       \ XOR AX,AX           \   NEG DI
   $29 C,  $D8 C,       \ SUB AX,BX           \   SBB EBX,EBX
   $1B C,  $DB C,       \ SBB BX,BX
endcode


EXEC: U<
code u<       ( m n -- f )
  $58 C,               \ pop ax
```

```
  $29 C, $D8 C,        \ sub ax,bx
  $1B C, $DB C,        \ sbb bx,bx
endcode



EXEC: C@  ( addr -- c )
code c@
  $8A C, $1F C,    \ mov bl,[bx]
  $32 C, $FF C,    \ xor bh,bh
endcode


EXEC: C!
code c!   ( c addr -- )
  $58 C,           \ pop ax
  $88 C, $07 C,    \ mov [bx],al
  mov([sp]+,tos)
endcode


EXEC: /MOD
code /mod   ( m n -- rem quot )
  $58 C,           \ pop ax
  $33 C, $D2 C,    \ xor dx,dx
  $F7 C, $F3 C,    \ div bx
  $8B C, $D8 C,    \ mov bx,ax
  $52 C,           \ push dx
endcode


EXEC: *
code mul  ( m n -- n )
  $58 C,           \ pop ax
  $F7 C, $E3 C,    \ mul bx
  $8B C, $D8 C,    \ mov bx,ax
endcode


\ --- i/o primitives ---

EXEC: ?KEY_DOS
code (?key)   ( -- f )
  mov(tos,-[sp])
  $B4 C, $0B C,    \ mov ah,0B
  $CD C, $21 C,    \ int 021
  $88 C, $C4 C,    \ mov ah,al
  $8B C, $D8 C,    \ mov bx,ax
endcode

EXEC: KEY_DOS
code key_dos     ( -- asc )
  mov(tos,-[sp])
  $B4 C, $08 C,    \ mov ah,8
  $CD C, $21 C,    \ int 021
```

```
  $8B C, $D8 C,     \ mov bx,ax
  $32 C, $FF C,     \ xor bh,bh
endcode

EXEC: EMIT_DOS
code emit_dos   ( asc -- )
  $B4 C, 02 C,      \ mov ah,2
  $89 C, $DA C,     \ mov dx,bx
  $CD C, $21 C,     \ int 021
  mov([sp]+,tos)
endcode

\ --- target primitive run time words ---
l: nobranch
  mov([sp]+,tos)
  mov([ip]+,wp)
endcode

DATA: 0BRANCH
code do0branch
  test(tos)
  nobranch bne           \ jne nobranch
  mov([sp]+,tos)
  add([ip],ip)
endcode

DATA: BRANCH
code dobranch
  add([ip],ip)
endcode

TASM
   : branch  dobranch  offs , ;
   : 0branch do0branch offs , ;
   : if      do0branch THERE 0 , ;
   : then    DUP offs NEGATE   SWAP >TARGET ! ;
   : else    dobranch THERE 0 , SWAP then  ;
   ' THERE   ALIAS begin
   ' branch  ALIAS again
   ' 0branch ALIAS until
   ' if      ALIAS while
   : repeat  SWAP branch then ;
TARGET

\ --- constants and variables ---
EXEC: TIB
$80 constant tib

EXEC: TRUE
TRUE constant true

EXEC: FALSE
FALSE constant false

EXEC: 0
```

```
0 constant zero

EXEC: 1
1 constant one

EXEC: 2
2 constant two

EXEC: BL
BL constant bl

EXEC: CELL
TCELL constant cell

EXEC: PLATFORM
TARGET_CPU constant platform

EXEC: STATE
variable state

EXEC: INPUTSTREAM
variable inputstream    0 ,

EXEC: LOADING
variable loading

EXEC: BUFSIZE           \ here to pad, behind pad, linebuffer
#132 constant bufsize

EXEC: VERSION
VERSION 1+  constant version


create rootuserdata
THERE
   0 ,                    \ link to next user   (user variable 2)
   #10 ,                  \ base                (user variable 1)
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
   0 ,
THERE

OVER -
EXEC: USIZE
```

```
    constant usize        \ bytes user space reserved for each user

EXEC: USER
create user             \ pointer to user space of active user
    ,

EXEC: UDP
create udp              \ incremented for each allocated user byte
   4 ,


EXEC: USERS
variable users            \ head of link through all users

EXEC: ROOT
: root  ( -- )
   rootuserdata
   user store ;

EXEC: NEXTUSER
: nextuser  ( -- addr )
   user fetch  ;


EXEC: BASE                \ user variable offset 2
: base ( -- addr )
   user fetch
   twoplus  ;

EXEC: BASE@
: base@  ( -- n )
  base fetch  ;

EXEC: BASE!
: base!  ( n -- )
  base store  ;

EXEC: CELL+
code cell+  ( n -- n )
                  inc(tos)
   TCELL 1 U> #IF  inc(tos)           #THEN
   TCELL 2 U> #IF  inc(tos)  inc(tos) #THEN
endcode


EXEC: CELL-
code cell-  ( n -- n )
                  dec(tos)
   TCELL 1 U> #IF  dec(tos)           #THEN
   TCELL 2 U> #IF  dec(tos)  dec(tos) #THEN
endcode


EXEC: CELL*
code cell*   ( n -- n )
```

```
  TCELL 1 U>  #IF  lsl(tos)  #THEN
  TCELL 2 U>  #IF  lsl(tos)  #THEN
endcode

EXEC: CELL/
code cell/   ( n -- n )
  TCELL 1 U>  #IF lsr(tos) #THEN
  TCELL 2 U>  #IF lsr(tos) #THEN
endcode


DATA: DOLIT
code dolit
  mov(tos,-[sp])
  mov([ip]+,wp)
  mov(wp,tos)
endcode



EXEC: COUNT
: count   ( addr -- addr c )
  oneplus dup
  oneminus c@  ;


TARGET_CPU #8086 = QUALIFIES
DATA: DOLITN
code dolitn
  mov(tos,-[sp])
  $AC C,           \ lodsb
  mov(ip,-[sp])
  $8B C, $D8 C,    \ mov bx,ax
  $32 C, $FF C,    \ xor bh,bh
  add(tos,ip)
endcode


TCELL 2 = QUALIFIES
DATA: DOARRAY
l: doarray
  lsl(tos)
  mov([sp]+,temp1)
  add(temp1,tos)
endcode


TARGET_CPU #8086 = QUALIFIES
DATA: DOVECTOR
l: dovector
  mov([sp]+,temp1)
  $FF C, $25 C,    \ JMP [DI]


"dolitn" FIND 0= QUALIFIES
```

```
DATA: DOLITN
: dolitn
  r> count
  2dup plus >r ;


"dovector" FIND 0= QUALIFIES
DATA: DOVECTOR
l: dovector
  docolon call
  swap fetch execute  ;




TASM
: lit       ( n -- )
  dolit  ,  ;

: litn      ( addr n -- )
  dolitn $,  ;

: array     ( n -- ) ( stream )
  comp:
  doarray call  ;

: vector    ( -- )   ( stream )
  comp:
  dovector call
  ;

: TLINK  ( addr -- )
  THERE SWAP
  EXCHANGE ,  ;


\ use TASM mem ops on target variables
: TVARIABLE  ( -- )
  HERE TASM CONSTANT
  TARGET  0 ,  ;

TARGET



EXEC: VECTORS
create vectors
TVARIABLE VECTORS



\ --- i/o vectors ---
EXEC: ?KEY
vector ?key
```

```
THERE (?key)
VECTORS TLINK
EXEC: >?KEY<
constant >?key<

EXEC: KEY
vector key
THERE key_dos
VECTORS TLINK
EXEC: >KEY<
constant >key<

EXEC: EMIT
vector emit
THERE emit_dos
VECTORS TLINK
EXEC: >EMIT<
constant >emit<


EXEC: PAUSE
vector pause
THERE noop
VECTORS TLINK
EXEC: >PAUSE<
constant >pause<



\ **v
1: romvoc                           \ mod: places to change marked with **v
   TVARIABLE TFIRST                 \ 0 FIRST
   TVARIABLE TLAST                  \ 1 LAST
THERE
   TVARIABLE TDP                    \ 2 DP
   TRUE ,     ( search flag )       \ 3 SEARCHING
   0 ,        ( allot count )       \ 4 ALLOTTED
   TRUE ,     ( case sensitive )    \ 5 CASESENSITIVE
   TVARIABLE TVOCNAME               \ 6 VOCNAME
   TVARIABLE TVOCFIND               \ 7 VOCFIND
   TVARIABLE TVOCCREATE             \ 8 VOCCREATE



EXEC: CONTEXT
variable context

EXEC: ROMDP
constant romdp

EXEC: #VOCS
   create #vocs  1 ,
```

```
EXEC: VOC
   array voc
   ( n= ) 8 HERE OVER CELL* +
   romvoc ,
   HERE - ALLOT0

EXEC: MAXVOCS
( n ) constant maxvocs



\ l: dovocarray ( -- addr )
\      mov([sp]+,temp1)            \ pop di
\      mov(tos,-[sp])             \ push bx
\      $8B C, $1E C,  ,           \ mov bx,[context]
\      lsl(tos)
\      add(temp1,tos)
\ endcode
\
\ : vocarray     ( n -- ) ( stream )
\      comp:
\      dovocarray call  ;
\
\
\ EXEC: FX
\    vocarray fx
\    4 CELL* ALLOT0
\
\

EXEC: FIRST
: first  ( -- addr )                    \ 0
   context fetch
   voc fetch   ;

EXEC: LAST
: last  ( -- addr )                     \ 1
   first cell+   ;

EXEC: DP
: dp   ( -- addr )                      \ 2
   last cell+   ;

EXEC: SEARCH
: search  ( -- addr )                   \ 3
   dp cell+   ;

EXEC: ALLOTTED
: allotted ( -- addr )                  \ 4
   search cell+  ;

EXEC: CASESENSITIVE
: casesensitive ( -- addr )             \ 5
\   allotted cell+  ;
   first #10 lit plus  ;
```

```
EXEC: VOCNAME
: vocname ( -- addr )                    \ 6
   casesensitive cell+  ;

EXEC: VOCFIND
: vocfind ( -- addr )                    \ 7
   vocname cell+  ;

EXEC: VOCCREATE
: voccreate ( -- addr )                  \ 8
   vocfind cell+  ;




EXEC: LATEST
: latest  ( -- addr )
   last fetch   ;


\ --- target hilevel ---


TARGET_CPU #8086 = QUALIFIES
EXEC: =
code equal  ( m n -- f )
  mov([sp]+,temp1)          \ POP DI
  sub(temp1,tos)            \ SUB BX,DI
   $83 C,  $EB C,  $01 C,   \ SUB BX,1
   $1B C,  $DB C,           \ SBB BX,BX
endcode

"equal" FIND 0= QUALIFIES
EXEC: =
: equal  ( m n -- f )
   notequal not   ;



TARGET_CPU #8086 = QUALIFIES
EXEC: 0=
code zeroequal
   $83 C,  $EB C,  $01 C,   \ SUB BX,1
   $1B C,  $DB C,           \ SBB BX,BX
endcode


"zeroequal" FIND 0= QUALIFIES
EXEC: 0=
: zeroequal  ( n -- f )
   zero equal   ;
```

```
TARGET_CPU #8086 = QUALIFIES
EXEC: 0<
code lesszero  ( n -- f )
   $D1 C, $E3 C,         \ SHL BX,1
   $1B C, $DB C,         \ SBB BX,BX
endcode

"lesszero" FIND 0= QUALIFIES
EXEC: 0<
: lesszero  ( n -- f )
   $8000 lit  u< not   ;



TARGET_CPU #8086 = QUALIFIES
EXEC: 0<>
code zeronotequal
   $33 C,  $C0 C,        \ XOR AX,AX
   $29 C,  $D8 C,        \ SUB AX,BX
   $1B C,  $DB C,        \ SBB BX,BX
endcode

"zeronotequal" FIND 0= QUALIFIES
EXEC: 0<>
: zeronotequal ( n -- f )
   0 <>  ;




TARGET_CPU #8086 = QUALIFIES
EXEC: SKIM
code skim
   $89 C, $DF C,           \ mov di,bx
   $83 C, $C3 C, CELL C,   \ add bx,2
   $53 C,                  \ push bx
   $8B C, $1D C,           \ mov bx,[di]
endcode

"skim" FIND 0= QUALIFIES
EXEC: SKIM
: skim    ( addr -- addr n )
  cell+ dup
  cell- fetch   ;




EXEC: CR
: cr   ( -- )
  ^M lit emit
  ^J lit emit   ;

EXEC: SPACE
: space   ( -- )
```

```
  bl emit   ;


EXEC: TYPE
: type ( addr n -- )
   ?dup if
     for
       count emit
     next
   then drop  ;


EXEC: TYPECR
: typecr ( addr n -- )
  2dup type
  plus oneminus c@       \ line continues (no CR) if last char is space
  bl notequal if cr then  ;

EXEC: SP
code sp  ( -- addr )
  mov(sp,temp1)
  mov(tos,-[sp])
  mov(temp1,tos)
endcode

EXEC: DEPTH
: depth  ( -- n )                              \ **stack
  sp0 fetch
  sp minus
  cell/
  oneminus   ;

EXEC: U>=
: u>=   ( m n -- f )
  u< not   ;

EXEC: U>
: u>   ( m n -- f )
  swap u<   ;

EXEC: 2@
: 2fetch  ( addr -- m n )
  skim swap fetch   ;

EXEC: 2!
: 2store  ( m n addr -- )
  tuck cell+
  store store   ;

: fix_9_a
  u>= 7 lit and
  '0 lit plus   ;

EXEC: CIPHER
: cipher   ( n -- c )
```

```
  dup #10 lit
  fix_9_a plus   ;

EXEC: DIGIT
: digit   ( c -- n )
  dup 'A lit
  fix_9_a minus   ;

EXEC: NUMERIC
: numeric ( asc -- f )
  '0 lit minus
  dup #17 lit u<
  if
    dup #10 lit u>= or
  else
    7 lit minus
  then
  base@ u<   ;


EXEC: UMIN     : umin      2dup u>   if  swap begin drop ;
EXEC: UMAX     : umax      2dup u< until swap  then drop ;



EXEC: ROT
code rot  ( n1 n2 n3 -- n2 n3 n1 )
   mov([sp]+,wp)
   mov([sp]+,temp1)
   mov(wp,-[sp])
   mov(tos,-[sp])
   mov(temp1,tos)
endcode


EXEC: -ROT
code -rot  ( n1 n2 n3 -- n3 n1 n2 )
   mov([sp]+,temp1)
   mov([sp]+,wp)
   mov(tos,-[sp])
   mov(wp,-[sp])
   mov(temp1,tos)
endcode



TARGET_CPU #8086 = QUALIFIES
EXEC: PLUCK
code pluck
   mov([sp]+,temp1)
   mov([sp]+,wp)
   mov(wp,-[sp])
   mov(temp1,-[sp])
   mov(tos,-[sp])
   mov(wp,tos)
```

```
endcode

"pluck" FIND 0= QUALIFIES
EXEC: PLUCK
: pluck  ( n1 n2 n3 -- n1 n2 n3 n1 )
  >r over
  r> swap   ;



TARGET_CPU #8086 = QUALIFIES
EXEC: 2SWAP
code 2swap  ( n1 n2 n3 n4 -- n3 n4 n1 n2 )
   mov([sp]+,wp)
   mov([sp]+,temp1)
   mov([sp]+,temp2)
   mov(wp,-[sp])
   mov(tos,-[sp])
   mov(temp2,-[sp])
   mov(temp1,tos)
endcode

"2swap" FIND 0= QUALIFIES
EXEC: 2SWAP
: 2swap  ( n1 n2 n3 n4 -- n3 n4 n1 n2 )
  rot  >r
  rot  r>   ;


TARGET_CPU #8086 = QUALIFIES
EXEC: 2OVER
code 2over ( n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2 )
   mov([sp]+,wp)
   mov([sp]+,temp1)
   mov([sp]+,temp2)
   mov(temp2,-[sp])
   mov(temp1,-[sp])
   mov(wp,-[sp])
   mov(tos,-[sp])
   mov(temp2,-[sp])
   mov(temp1,tos)
endcode

"2over" FIND 0= QUALIFIES
EXEC: 2OVER
: 2over  ( n1 n2 n3 n4 -- n1 n2 n3 n4 n1 n2 )
  >r >r 2dup
  r> r> 2swap   ;



EXEC: EXCHANGE
code exchange ( n addr -- n )
   mov([sp]+,temp1)
   swap(temp1,[tos])
```

```
   mov(temp1,tos)
endcode



TARGET_CPU #8086 = QUALIFIES
EXEC: CSET
code cset   ( c addr -- )
   $5A C,          \ pop dx
   $08 C, $17 C,   \ or [bx],dl
   $5B C,          \ pop bx
endcode

"cset" FIND 0= QUALIFIES
EXEC: CSET
: cset   ( c addr -- )
   >r r@ c@ or r> c!   ;




TARGET_CPU #8086 = QUALIFIES
EXEC: CRESET
code creset   ( c addr -- )
   $5A C,          \ pop dx
   $F6 C, $D2 C,   \ not dx
   $20 C, $17 C,   \ and [bx],dl
   $5B C,          \ pop bx
endcode

"creset" FIND 0= QUALIFIES
EXEC: CRESET
: creset   ( c addr -- )
   >r not r@ c@ and r> c!   ;







EXEC: -RIGHT$
: -right$  ( addr n n -- addr n )
  over umin minus   ;


: leading  ( addr n c -- addr n f )
  pluck c@ equal   ;


: lastchar ( addr cnt -- asc )
   plus oneminus c@  ;

EXEC: -TRAILING
: -trailing  ( addr cnt -- addr cnt )
  begin
```

```
    dup dup if
      drop
      2dup lastchar
      bl equal
    then
  while
    oneminus
  repeat   ;


EXEC: -LEFT$
: -left$  ( addr n n -- addr n )
  over umin
  tuck minus
  -rot plus swap   ;

EXEC: SCAN
: scan ( addr n c -- addr n )
  >r
  begin
    over c@
    r@ notequal
    over and
  while
    one -left$
  repeat
  r> drop   ;

EXEC: SKIP
: skip   ( addr n c -- addr n )
  >r
  begin
    over c@
    r@ equal
    over and
  while
    one -left$
  repeat
  r> drop   ;

EXEC: TEXT
: text  ( addr n c -- addr n addr n )
  dup >r skip
  r> pluck >r scan
  r> pluck
  over minus   ;

EXEC: !STREAM
: !stream  ( addr cnt -- )
   inputstream 2store  ;

EXEC: @STREAM
: @stream  ( -- addr cnt )
   inputstream 2fetch  ;
```

```
EXEC: STREAM
: stream ( asc -- addr n )
  @stream
  rot text 2swap
  !stream  ;

EXEC: -STREAM
: -stream ( n -- )
  @stream
  rot -left$
  !stream  ;

EXEC: WORD
: word  ( -- addr n )
  bl stream   ;

EXEC: \ IMMEDIATE
: backslash  ( -- )
  zero stream 2drop   ;

EXEC: ( IMMEDIATE
: comment  ( -- )
  ') lit stream 2drop
  one -stream   ;


EXEC: ?LEAVE
: ?leave   ( f -- )
  if leave then   ;



EXEC: SWITCHES
create switches
TVARIABLE SWITCHES




TARGET_CPU #8086 = QUALIFIES
EXEC: MATCH$
code match$  ( addr n addr n -- f )
   $1E C,                \ PUSH DS
   $07 C,                \ POP ES
   $5F C,                \ pop     di
   $58 C,                \ pop     ax
   $5A C,                \ pop     dx
   $51 C,                \ push    cx
   $33 C, $C9 C,         \ XOR     CX,CX
   $87 C, $CB C,         \ XCHG    BX,CX
   $3B C, $C1 C,         \ CMP     AX,CX
   $75 C, $0B C,         \ JNE     MIS1
   $E3 C, $09 C,         \ JCXZ    MIS1
   $56 C,                \  push    si
```

```
   $89 C, $D6 C,          \  mov     si,dx
   $F3 C, $A6 C,          \  REPZ    CMPSB
   $75 C, $01 C,          \  JNE     MIS2
   $4B C,                 \  DEC     BX
                          \ mis2:
   $5E C,                 \  pop     si
                          \ mis1:
   $59 C,                 \ pop     cx
endcode


"match$" FIND 0= QUALIFIES
EXEC: MATCH$
: match$  ( addr n addr n -- f )
  rot over equal
  dup if drop
    begin
      dup
    while
      >r >r count
      r> count
      rot minus
      r> swap if
        3drop
        false exit
      then oneminus
    repeat  true
  then
  nip nip nip   ;


EXEC: UPPERCASE
: uppercase ( asc -- asc )
   dup  'a lit u< not
   over 'z 1+ lit u< and
   if
      'a 'A - lit minus
   then ;

EXEC: MATCHANYCASE$
: matchanycase$ ( addr n addr n -- f )
  rot over equal
  dup if drop
    begin
      dup
    while
      >r >r count uppercase
      r> count uppercase
      rot minus
      r> swap if
        3drop
        false exit
      then oneminus
    repeat  true
  then
```

```
  nip nip nip   ;



\ --------------- dictionary search ----------------

variable vocseg

EXEC: NAME                                          \ **still
: name   ( addr -- addr n )
  cell+ count
  #31 lit and   ;

EXEC: (FIND)       THERE                                 \ **still
: (find)  ( addr n -- addr | 0 )
  false -rot
  first fetch >r
  latest
  begin
    dup r@ notequal
  while
    name 2over 2over
    casesensitive fetch
    if
       match$
    else
       matchanycase$
    then
    if
      drop cell- oneminus
      nip nip nip
      dup false r@
    else
      plus
    then
    pause
  repeat
  r> 2drop 2drop   ;
TVOCFIND !


EXEC: INVOC
variable invoc

EXEC: (VOCFIND)                           \ **ok
: (vocfind)    ( addr n -- addr | 0 )
  context fetch >r
  zero
  #vocs fetch for
    i oneminus context store
    search fetch if
      drop
      2dup
      vocfind fetch execute
```

```
      dup if
        context fetch
        invoc store
        leave
      then
    then
  next nip nip
  r> context store   ;


EXEC: FIND                              \ **ok
vector find   ( addr n --addr | 0 )
THERE (vocfind)
VECTORS TLINK
EXEC: >FIND<
constant >find<




#64 constant imm
#32 constant exec

: headerflag ( mask -- )                        \ **still
  latest cell+ creset   ;

: headerflag? ( addr mask -- f )
  swap cell+
  c@ and
  zeroequal   ;

EXEC: EXECUTABLE?
: executable?  ( addr -- flag )
  exec headerflag?   ;

EXEC: EXECUTABLE
: executable   ( -- )
  exec headerflag   ;

EXEC: IMMEDIATE?
: immediate?   ( addr -- flag )
  imm headerflag?   ;

EXEC: IMMEDIATE
: immediate   ( -- )
  imm headerflag   ;


\ ---------------------                   ----------------------



: nakednumber?  ( addr cnt -- f )
  true swap
  ?dup if
```

```
    for
      >r count
      numeric
      r> and
      dup zeroequal ?leave
    next
  then nip   ;

EXEC: (NUMBER?)
: (number?)   ( addr n -- f )
  dup if
    '# lit leading
    if
      #10 lit
    else
      '$ lit leading
      if
        $10 lit
      else
        '% lit leading
        if
          %10 lit
        else
          nakednumber? exit
        then
      then
    then
    base exchange >r
    one -left$ nakednumber?
    r> base!
  else
    nip
  then   ;


: nakednumber
  zero swap
  ?dup if
    for
      base@ mul >r
      count digit r> plus
    next
  then nip   ;

EXEC: (NUMBER)
: (number)   ( addr cnt -- n )
  '# lit leading
  if
    #10 lit
  else
    '$ lit leading
    if
      $10 lit
    else
      '% lit leading
```

```
      if
        %10 lit
      else
        nakednumber exit
      then
    then
  then
  base exchange >r
  one -left$ nakednumber
  r> base!   ;


EXEC: NUMBER?
vector number?
THERE (number?)
VECTORS TLINK
EXEC: >NUMBER?<
constant >number?<


EXEC: NUMBER
vector number
THERE (number)
VECTORS TLINK
EXEC: >NUMBER<
constant >number<


EXEC: ERROR
vector error
THERE  return_to_dos
VECTORS TLINK
EXEC: >ERROR<
constant >error<


EXEC: ?ERROR
: ?error   ( f addr n -- )
  rot if
    error
  then 2drop   ;


EXEC: <-
code swapstore  ( addr n -- )
   mov(tos,temp1)
   mov([sp]+,tos)
   swap(temp1,[tos])
   mov([sp]+,tos)
endcode


TARGET_CPU #8086 = QUALIFIES
EXEC: +!
code plusstore  ( n addr -- )
```

```
        $58 C,              \ pop     ax
        $01 C, $07 C,       \ add [bx],ax
        $5B C,              \ pop bx
endcode

"plusstore" FIND 0= QUALIFIES
EXEC: +!
: plusstore  ( n addr -- )
        tuck fetch
        plus swapstore  ;


EXEC: OFF
: off   ( addr -- )
  false swapstore   ;


EXEC: ON
: on    ( addr -- )
  true swapstore   ;



TARGET_CPU #8086 = QUALIFIES
EXEC: INC
code inc ( addr -- )
        $FF C, $07 C,   \ inc w[bx]
        $5B C,          \ pop bx
endcode


"inc" FIND 0= QUALIFIES
EXEC: INC
: inc   ( addr -- )
  one swap plusstore   ;




TARGET_CPU #8086 = QUALIFIES
EXEC: DEC
code dec ( addr -- )
        $FF C, $0F C,   \ dec w[bx]
        $5B C,          \ pop bx
endcode


"inc" FIND 0= QUALIFIES
EXEC: INC
: inc   ( addr -- )
  one swap plusstore   ;
```

```
EXEC: HERE      THERE
: here   ( -- addr )
  dp fetch  ;  ALIAS BEGIN IMMEDIATE


: toffs
  here minus  ;

EXEC: FREE
: free     ( -- n )                      \ **still
  latest
  bufsize minus
  toffs  ;



EXEC: ALLOT
: allot ( n -- )
  dup $20 lit plus
  free u>
     "voc full" litn ?error
  dup dp plusstore
  allotted plusstore  ;


EXEC: -ALLOT
: -allot ( addr -- )
  dp store   ;


EXEC: ,
: comma  ( n -- )
  here
  cell allot
  store  ;

EXEC: C,
: ccomma  ( c -- )
  here
  one allot
  c!  ;

\ code type dependent
EXEC: A,
: acomma  ( n -- )
  comma  ;



EXEC: DOLITN,
: dolitn,  ( -- )
   dolit dolitn acomma  ;

EXEC: DOLIT,
: dolit,  ( -- )
```

```
   dolit dolit acomma  ;


EXEC: COMPILING
: compiling  ( -- f )
  state fetch   ;

EXEC: LITERAL IMMEDIATE
: literal ( n -- | n )
  compiling
  if
    dolit, comma
  then   ;

EXEC: <CMOVE
: <cmove   ( src dest n -- )
  ?dup if
    for
      >r count
      r@ c!
      r> oneplus
    next
  then 2drop   ;


EXEC: CMOVE>
: cmove> ( src dest n -- )
  ?dup if
    dup >r plus
    swap r@ plus
    swap r> for
      oneminus swap
      oneminus tuck
      c@ over c!
    next
  then 2drop   ;

EXEC: CMOVE
: cmove   ( src dest n -- )
  pluck pluck
  u< if
    cmove>
  else
    <cmove
  then   ;


EXEC: MOVE$
: move$  ( src n dest -- )
  swap $FF lit and
  swap 2dup c! oneplus
  swap cmove   ;

EXEC: $,
```

```
: str,  ( addr n --)
  here over oneplus
  allot move$   ;


\ --- prefix extension ---
\ adding:  -123  'A   ^G   "string"
EXEC: WILDSTR
: wildstring ( addr n -- )
  2dup lastchar
  '" lit notequal if     ( string contains spaces )
    '" lit stream
    one -stream
    nip plus oneplus
  then
  one -left$
  one -right$
  compiling if
    dolitn,  str,
  then   ;



: firstchar  ( addr n mask -- )
  -rot dup two notequal if error then
  drop oneplus c@ and literal
    ;


EXEC: WILDASC
: wildascii ( addr n -- )
  $FF lit firstchar  ;

EXEC: WILDCONTROL
: wildcontrol ( addr n -- )
  BL 1- lit firstchar ;


EXEC: WILDNUMBER
: wildnumber ( addr n -- )
  2dup one -left$ number?
  if
    one -left$ number
    negate literal
  else
    error
  then   ;


EXEC: WILDCARDS
create wildcards  4 ,
  '^ ,  wildcontrol          \ ^M  =  CONTROL M  =  $0D
  '' ,  wildascii            \ 'A  =  ASCII A  =  $41
  '- ,  wildnumber           \ -20  =  20 NEGATE
  '" ,  wildstring           \ "these kind of strings"
```

```
   0 ,   0 ,                  \ >DOSCMD
   0 ,   0 ,                  \ =n filenumber
   0 ,   0 ,
   0 ,   0 ,
   0 ,   0 ,
   0 ,   0 ,
   0 ,   0 ,
   0 ,   0 ,
\ should warn when full


EXEC: LOOKUP
: lookup  ( n addr -- n | 0 )
  false -rot
  skim
  ?dup if
    for
      skim pluck equal
      if
        fetch -rot
        leave
      then
      cell+
    next
  then
  2drop   ;



: wildies
  over c@                              \ first char of unknown string
  wildcards lookup                     \ pattern defined ?
  ?dup if
    execute
  else
    error                             \ can't dig
  then   ;


EXEC: UNKNOWN
vector unknown
THERE  wildies
VECTORS TLINK
EXEC: >UNKNOWN<
constant >unknown<

EXEC: ?STACK
: ?stack                                                \ **stack
  depth dup lesszero       "stack underflow" litn ?error
  dssize fetch cell/  u>=  "stack overflow"  litn ?error   ;


: do0branch,
  dolit do0branch comma   ;
```

```
: dobranch,
  dolit dobranch  comma   ;



EXEC: MARK>  IMMEDIATE
: mark>
  here zero comma   ;

EXEC: <RESOLVE   IMMEDIATE
:  <resolve
  dup toffs negate swapstore   ;



EXEC: UNTIL IMMEDIATE
: tuntil                        \ ( addr -- ) compile time
  do0branch,                    \ ( -- )      run time
  toffs comma   ;

EXEC: NEVER IMMEDIATE
: tnever                        \ ( -- addr ) compile time
  dobranch, mark>  ;            \ ( -- )      run time

EXEC: IF  IMMEDIATE
THERE
: tif                           \ ( -- addr ) compile time
  do0branch, mark>   ;          \ ( f -- )    run time
ALIAS WHILE IMMEDIATE

EXEC: THEN IMMEDIATE
: tthen                         \ ( addr -- ) compile time
  <resolve  ;                   \ ( -- )      run time

EXEC: ELSE IMMEDIATE
: telse
  tnever
  swap <resolve  ;

EXEC: AGAIN IMMEDIATE
: tagain
  dobranch, toffs comma  ;

EXEC: REPEAT IMMEDIATE
: trepeat
  swap
  tagain
  <resolve  ;



EXEC: CALL,
: call,
  $E8 lit ccomma
```

```
  here cell+ minus comma   ;

EXEC: '
: tick     ( <stream> -- addr )
  word 2dup find
  ?dup if nip nip fetch
  else error then   ;

EXEC: [COMPILE] IMMEDIATE
: [compile]  ( <stream> -- )
  tick acomma   ;

EXEC: ?COMPILE
: ?compile   ( addr -- )
  compiling if acomma
  else execute then   ;

EXEC: [   IMMEDIATE
: leftbracket  ( -- )
  state off   ;

EXEC: ]
: rightbracket  ( -- )
  state on   ;

EXEC: FILL
: fill    ( addr n c -- )
  -rot
  ?dup if
    for
      2dup c!
      oneplus
    next
  then 2drop   ;

EXEC: ERASE
: erase   ( addr n -- )
  zero fill   ;


DATA: WARNINGS
1: warnings
   TRUE ,
   SWITCHES TLINK


EXEC: WARNING
: warning   ( addr n -- )
  warnings lit fetch if
    "warning: " litn type
    2dup typecr
  then 2drop   ;


EXEC: ?WARNING
```

```
: ?warning  ( f addr n -- )
   rot if
      2dup warning
   then  2drop  ;


EXEC: PAD
: pad  ( -- addr )
   romdp fetch
   bufsize plus  ;


EXEC: BUILD_HEADER
: build_header  ( addr u addr -- )
    here over store cell+
    over %11100000 lit or
    over c! oneplus                   \ name count + header flags
    swap cmove  ;


EXEC: (CREATE)       THERE                \ **still
: (create)  ( addr n -- )
  dup if
    #31 lit umin
    warnings lit fetch if
       2dup find if
          "redef " litn warning
          2dup typecr
       then
       "(create): " litn type
    then

    2dup latest                           \ addr n addr n addr
    over oneplus cell+ minus              \ addr n addr n addr-
    dup last store                        \ addr n addr n addr-
    build_header
  then 2drop   ;
TVOCCREATE !

EXEC: (VOCCREATE)
: (voccreate)  ( addr u -- )
        voccreate fetch execute ;

vector ((create))
THERE  (voccreate)
VECTORS TLINK
EXEC: >CREATE<
constant >create<


EXEC: CREATE
: tcreate ( stream -- )
  word ((create))  ;
```

```
: (semierror)
  "unsolicited semicolon" litn error   ;


EXEC: ; IMMEDIATE
vector dosemicolon
THERE  (semierror)
constant >;<


EXEC: COMPILED
: compiled   ( addr -- )
  >;< store   ;

EXEC: SEMIERROR
: semierror   ( -- )
   dolit (semierror) compiled   ;


EXEC: NEST,
: nest,
  docolon lit call,   ;

EXEC: UNNEST,
: unnest,
  dosemi lit acomma   ;

: ;does
  unnest,
  leftbracket
  semierror   ;

DATA: (;)
: (semicolon)
  ;does executable   ;

EXEC: HILEVEL
: hilevel   ( end_of_compilation_action -- )
  nest,
  rightbracket
  compiled   ;

EXEC: :
: tcolon
  tcreate
  dolit (semicolon) hilevel   ;

EXEC: CONSTANT
: tconstant
  tcreate executable
  doconst lit call, comma   ;

EXEC: ALIAS
: talias                                \ **still
  tcreate executable
```

```
  latest store   ;

EXEC: NEXT,
: next,
  dolitn  HERE 0 C, endcode
  HERE OVER 1+ - SWAP C!
  here swap dup allot cmove   ;

EXEC: DOES:
: does:   ( -- addr )
  here
  dolit ;does hilevel   ;


: (build)
  tcreate executable
  $4D lit ccomma    \ dec_bp
  $4D lit ccomma    \ dec_bp
  $89 lit ccomma    \
  $76 lit ccomma      \
  $00 lit ccomma        \ mov [bp],si
  $BE lit ccomma    \
  here cell allot     \ mov si,nnnn
  $E9 lit ccomma    \ jmp addr
  r> skim
  here cell+ minus
  comma  >r
  here swapstore ;    \ write addr to nnnn in mov si,nnnn


EXEC: BUILD IMMEDIATE
: build
  dolit (build) acomma comma   ;


\ don't de-allot with ALLOT0 ! Rather use ALLOT .
EXEC: ALLOT0
: allot0  ( n -- )
  here over erase allot   ;


: setup_voc ( size n -- )       \ **v   \ **still
  here rot allot0
  swap voc
  here tuck swapstore
  dup comma comma comma     \ first last dp
  false comma               \ search
  zero comma                \ allotted
  true comma                \ case sensitive
  latest fetch comma        \ body (for name id)
  dolit (find) comma        \ default find
  dolit (create) comma  ;   \ default create


: (dovoc)
```

```
  r> c@ context store
  search on   ;

EXEC: VOCABULARY
: vocabulary ( size <stream> -- )
  (build) (dovoc)
  #vocs
    tuck fetch
    dup ccomma
    setup_voc
  inc   ;


EXEC: ROM
THERE TVOCNAME !
: rom   (dovoc) 0 C,


create (screensize)
   #80 ,  #25 ,

EXEC: SCREENSIZE
: screensize  ( -- cols lines )
   (screensize) 2fetch  ;

EXEC: !SCREENSIZE
: !screensize  ( cols lines -- )
   (screensize) 2store  ;

EXEC: LINELEN
: linelen  ( -- n )
   screensize drop oneminus  ;


EXEC: LINECOUNTER
variable linecounter


EXEC: CTRLKEY
array ctrlkey
   noop     noop     noop     noop     noop     noop     noop     noop
   noop     noop     noop     noop     noop     noop     noop     noop
   noop     noop     noop     noop     noop     noop     noop     noop
   noop     noop     noop     noop     noop     noop     noop     noop

EXEC: PERFORM
: perform  ( addr -- )
   fetch execute  ;


EXEC: (EXPECT)
: (expect) ( addr max -- n )
  @stream >r >r
  zero swap                              \ addr n max
  begin
```

```
    key                                 \ addr n max asc
    dup bl u<
    if
      swap >r >r !stream
      r@ ctrlkey perform ?stack
      @stream r> r> swap                \ addr n max asc
    then
    dup ^M lit minus
  while                                 \ addr n max asc
    dup bl u< not
    if
      swap >r                           \ addr n asc              max
      over r@ u<                        \ addr n asc f
      swap >r                           \ addr n f                max asc
      if                                \ addr n                  max asc
        r@ emit                         \ addr n                  max asc
        2dup plus                       \ addr n addr             max asc
        r@ swap c! oneplus              \ addr n                  max asc
      then
      r> r> swap                        \ addr n max asc
    then drop                           \ addr n max
  repeat
  2drop nip
  r> r> !stream  ;



EXEC: EXPECT
vector expect
THERE (expect)
VECTORS TLINK
EXEC: >EXPECT<
constant >expect<


EXEC: (QUERY)
: (query)  ( -- addr n )
  tib dup
  linelen bufsize umin
  2dup erase expect
  linecounter inc  ;


EXEC: QUERY
vector query
THERE (query)
VECTORS TLINK
EXEC: >QUERY<
constant >query<


EXEC: RELOCATE-HEADERS
: relocate-headers ( dest -- )        \ **still
  dup last exchange
  tuck 2dup minus
```

```
  first fetch -rot
  first plusstore
  minus cmove  ;


EXEC: NEXTWORD
: nextword  ( -- addr cnt )         \ like WORD but crosses end of line
  @stream -trailing
  zeroequal if                      \ end of line
    query !stream                   \ read next
  then drop word   ;

EXEC: #SKIP
: #skip ( -- )
  5 lit for  ">" litn typecr  next
  dolit drop  >emit< exchange
  begin
    nextword
    2dup "#THEN" litn match$
  not while
    "#IF" litn match$
    if #skip then
  repeat 2drop
  >emit< store   ;

EXEC: #IF
: #if      ( f -- )
  zeroequal if
    #skip
  then   ;

EXEC: #THEN
code #then  endcode

EXEC: QUALIFIES
: qualifies ( f -- )
  zeroequal if
    backslash
    begin
      query nip
    zeroequal until
  then  ;


EXEC: INTERPRETER
: interpreter ( ??? addr cnt -- ??? )
  2dup find
  ?dup if  ( header found )
    nip nip
    dup fetch swap
    dup executable?
    if
      immediate? if
        execute
      else
```

```
        ?compile
      then
    else
      drop literal
    then
  else
    2dup number?
    if
      number literal
    else
      unknown
    then
  then
  ?stack   ;

EXEC: INTERPRET
: interpret  ( addr n -- )
  -trailing !stream
  begin
    @stream nip
  while
    word interpreter
  repeat ;



EXEC: PROMPT
vector prompt
THERE  noop
VECTORS TLINK
EXEC: >PROMPT<
constant >prompt<

EXEC: QUIT
: quit  ( ? -- )
  initstacks
  leftbracket
  begin
    query
    interpret
    prompt
  again   ;


EXEC: INFILE
create infile
#68 ALLOT0

EXEC: FILENAME
: filename  ( file -- addr n )
  cell+ cell+ count   ;

EXEC: FILENM
: filenm ( addr len h -- )
  cell+ cell+
```

```
  dup #64 lit erase
  move$   ;


EXEC: NAMEFILE
: namefile ( file <stream> -- )
  word rot filenm   ;


EXEC: INT21
code int21 ( ax bx cx dx -- ax bx cx dx carry )
  $8B C, $F9 C, \ mov di,cx
  $89 C, $DA C, \ mov dx,bx
  $59 C,        \ pop cx
  $5B C,        \ pop bx
  $58 C,        \ pop ax
  $CD C, $21 C, \ int 021
  $50 C,        \ push ax
  $53 C,        \ push bx
  $51 C,        \ push cx
  $52 C,        \ push dx
  $1B C, $DB C, \ sbb bx,bx     \ carry -> bx  (0,-1)
  $8B C, $CF C, \ l2: mov cx,di
endcode


: fileservice  ( h n -- err )
  over zero dup
  rot 5 lit plus
  int21
  nip nip nip
  not if   ( handle )
    over dup cell+ on
    store zero
  then nip   ;


EXEC: OPEN-FILE
: open-file   ( file -- err )
  $3D02 lit fileservice   ;

EXEC: NEW-FILE
: new-file    ( file -- err )
  $3C00 lit fileservice   ;


EXEC: CLOSE-FILE
: close-file ( file -- err )
  skim $3E00 lit swap
  zero dup
  int21 nip nip nip
  and tuck swapstore   ;


EXEC: ?FILEERROR
```

```
: ?fileerror  ( err -- )
  "*FILE*" litn ?error   ;


EXEC: USEFILE
: usefile  ( file <stream> -- )
  dup namefile
  open-file
  ?fileerror   ;


EXEC: R/W
code r/w ( handle length seg buf command -- results.. )
  $4D C,                  \     dec bp
  $4D C,                  \     dec bp
  $8C C, $5E C, $00 C,    \     mov [bp],ds
  $8B C, $F9 C,           \     mov di,cx
  $8A C, $E3 C,           \     mov ah,bl
  $5A C,                  \     pop dx
  $1F C,                  \     pop ds
  $59 C,                  \     pop cx
  $5B C,                  \     pop bx
  $2E C, $8B C, $1F C,    \     mov cs:bx,[bx]
  $CD C, $21 C,           \     int 021
  $8B C, $D8 C,           \     mov bx,ax
  $72 C, $03 C,           \     jc >L1
  $53 C,                  \      push bx
  $33 C, $DB C,           \      xor bx,bx
  $8E C, $5E C, $00 C,    \ L1: mov ds,[bp]
  $45 C,                  \     inc bp
  $45 C,                  \     inc bp

  $8B C, $CF C,           \     mov cx,di
endcode

EXEC: LREAD-FILE
: lread-file ( handle # seg addr -- n 0 | errorcode )
  #63 lit r/w
  2dup or
  zeroequal
  if
    drop not
  then   ;

EXEC: LWRITE-FILE
: lwrite-file ( handle # seg addr -- n 0 | err )
  #64 lit r/w   ;

EXEC: DSEG
code dseg   ( -- ds )
  mov(tos,-[sp])
  $8C C, $DB C,   \ MOV BX,DS
endcode

EXEC: READ-FILE
```

```
: read-file  ( handle # addr -- n 0 | errorcode )
  dseg swap lread-file   ;

EXEC: WRITE-FILE
: write-file  ( h n addr -- n 0 | err )
  dseg swap lwrite-file   ;


: bputfileh ( adr n h -- )
  dup 2swap swap write-file
  ?fileerror drop close-file   ;


: bsaveh ( adr1 n  h -- f )
  dup new-file
  ?fileerror
  bputfileh   ;

EXEC: BSAVE
: bsave  ( adr1 n -- f )
  infile namefile
 infile bsaveh   ;




EXEC: #SOURCEFILES
variable #sourcefiles

EXEC: SOURCEFILE
variable sourcefile

THERE "METASTIC.MS'" $,
EXEC: SOURCEFILENAMES
array sourcefilenames
  ,
  MAXSOURCEFILES 1- TCELL * ALLOT0


: moresourcefiles ( -- f )
  #sourcefiles fetch
  MAXSOURCEFILES lit u<  ;


: ?sourcefile ( addr n -- handle true | addr n false )
  false -rot
  #sourcefiles fetch
  ?dup if
     for
        2dup
        i sourcefilenames fetch count
        match$ if
          i -rot
          leave
        then
```

```
     next
  then
  pluck if  ( 0 h addr cnt )
     2drop nip true
  else      ( 0 addr cnt )
     rot
  then  ;


EXEC: NEWSOURCEFILE
: newsourcefile ( addr cnt -- )
   moresourcefiles if
     ?sourcefile   ( unusual stack ! )
     if
        sourcefile store
     else
        #sourcefiles inc
        here #sourcefiles fetch
        dup sourcefile store
        sourcefilenames store
        str,
     then
   else
      2drop
   then
    ;



\ addr: pointer to next chain element. For executing linked list code.
EXEC: CHAINRUN
: chainrun ( addr -- )
  fetch
  begin
    ?dup
  while
    skim swap execute
  repeat   ;



EXEC: INITSTUFF
variable initstuff

EXEC: EXITSTUFF
variable exitstuff


EXEC: RETURN
: return ( n -- )
   exitstuff chainrun
   return_to_dos ;

EXEC: BYE
```

```
: bye
   zero return  ;


EXEC: SAVE-SYSTEM
: save-system  ( <stream> -- )                  \ **still
  rom
  exitstuff chainrun
  pad relocate-headers    \ minimize space between HERE and headers
  PROGSTART lit first fetch
  over minus bsave                \ save code + headers
  ?fileerror
  zero return_to_dos  ;



EXEC: STRETCH                                    \ **still
: stretch  ( -- )                                \ **stack
  dsegsize fetch
  #16 lit mul                           \ convert paragraphs to bytes
  first fetch minus
  latest plus
  relocate-headers ;


EXEC: INTERACTIVE
: interactive ( -- )
   @stream interpret
   linecounter off
   prompt quit   ;


vector start
THERE interactive
VECTORS TLINK
EXEC: >START<
constant >start<


: (boot)
   stretch
   tib count
   bl skip
   !stream
   initstuff chainrun
   start
    ;


vector boot
THERE (boot)
VECTORS TLINK
EXEC: >BOOT<
constant >boot<
```

```
\ --- target main program ---
byevector offs NEGATE
byevector TCELL - >TARGET !

  $89 C, $E5 C,              \ MOV BP,SP
  mov(ip,-[sp])
: restart
  initstacks
  dseg vocseg store
  boot bye   ;

WARNINGS ON

\ --- write target image ---
FIRST @ T0 -  TFIRST !
LATEST T0 -   TLAST !
THERE         TDP !
SAVE-TARGET METASTIC.COM
```