4e4th Glossary. Release0.34 - Build 2012-04-11
http://www.forth-ev.de/repos/4e4th/

Alphabetical Order

```
;C  !            x a-addr --                    store cell in memory
;Z  !CF          adrs cfa --                    set code action of a word
;Z  !COLON       --                             change code field to DOCOLON
;Z  !DEST        dest adrs --                   change a branch dest'n
;C  #            ud1 -- ud2                     convert 1 digit of output
;C  #>           ud1 -- c-addr u                end conv., get string
;C  #S           ud1 -- ud2                     convert remaining digits
;Z  #init        -- n                           #bytes of user area init data
;C  '            -- xt                          find word in dictionary
;Z  'source      -- a-addr                      two cells: len, adrs
;C  (            --                             skip input until )
;Z  (+loop)      n -- R: sys1 sys2 -- | sys1 sys2run-time code for +LOOP
;Z  (DOES>)      --                             run-time action of DOES>
;Z  (IS")        -- c-addr u                    run-time code for S"
;Z  (S")         -- c-addr u                    run-time code for S"
;U  (crc         n addr len -- n'               crc process string including previous crc-byte
;Z  (do)         n1|u1 n2|u2 -- R: -- sys1 sys2run-time code for DO
;Z  (loop)       R: sys1 sys2 -- | sys1 sys2    run-time code for LOOP
;C  *            n1 n2 -- n3                    signed multiply
;C  */           n1 n2 n3 -- n4                 n1*n2/n3
;C  */MOD        n1 n2 n3 -- n4 n5              n1*n2/n3, rem&quot
;C  +            n1/u1 n2/u2 -- n3/u3           add n1+n2
;C  +!           n/u a-addr --                  add cell to memory
;C  +LOOP        adrs -- L: 0 a1 a2 .. aN --    finish a loop
;C  ,            x --                           append cell to dict
;Z  ,BRANCH      xt --                          append a branch instruction
;Z  ,CALL        adrs --                        append a subroutine CALL
;Z  ,CF          adrs --                        append a code field
;Z  ,DEST        dest --                        append a branch address
;Z  ,EXIT        --                             append hi-level EXIT action
;Z  ,JMP         adrs --                        append an absolute 16-bit JMP
;Z  ,NONE        --                             append a null destination (Flashable)
;C  -            n1/u1 n2/u2 -- n3/u3           subtract n1-n2
;C  .            n --                           display n signed
;C  ."           --                             compile string to print
;U  .COLD        --                             display COLD message
;X  .S           --                             print stack contents
;Z  .VER         --                             type message
;C  /            n1 n2 -- n3                    signed divide
;C  /MOD         n1 n2 -- n3 n4                 signed divide/rem'dr
;X  /STRING      a u n -- a+n u-n               trim string
;C  0<           n -- flag                      true if TOS negative
;C  0=           n/u -- flag                    return true if TOS=0
;C  1+           n1/u1 -- n2/u2                 add 1 to TOS
;C  1-           n1/u1 -- n2/u2                 subtract 1 from TOS
;U  1MS          --                             wait about 1 millisecond
;C  2!           x1 x2 a-addr --                store 2 cells
;C  2*           x1 -- x2                       arithmetic left shift
;C  2/           x1 -- x2                       arithmetic right shift
;C  2@           a-addr -- x1 x2                fetch 2 cells
;C  2CONSTANT    --                             define a Forth double constant
;C  2DROP        x1 x2 --                       drop 2 cells
;C  2DUP         x1 x2 -- x1 x2 x1 x2           dup top 2 cells
;C  2OVER        x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2
;C  2SWAP        x1 x2 x3 x4 -- x3 x4 x1 x2     per diagram
;C  :            --                             begin a colon definition
;C  ;            --                             end a colon definition
;U  ;PN          --                             send delimiter ; followed by parameter
;C  <            n1 n2 -- flag                  test n1<n2, signed
;C  <#           --                             begin numeric conversion
;X  <>           x1 x2 -- flag                  test not eq (not ANSI)
;Z  <BUILDS      --                             define a word with t.b.d. action & no data
;C  =            x1 x2 -- flag                  test x1=x2
;C  >            n1 n2 -- flag                  test n1>n2, signed
;Z  ><           x1 -- x2                       swap bytes (not ANSI)
;C  >BODY        xt -- a-addr                   adrs of CREATE data
;C  >IN          -- a-addr                      holds offset into TIB
;Z  >L           x -- L: -- x                   move to leave stack
;C  >NUMBER      ud adr u -- ud' adr' u'        convert string to number
;C  >R           x -- R: -- x                   push to return stack
;Z  >counted     src n dst --                   copy to counted str
```

```
;Z  >digit       n -- c                      convert to 0..9A..Z
;Z  ?ABORT       f c-addr u --               abort & print msg
;Z  ?DNEGATE     d1 n -- d2                  negate d1 if n negative
;C  ?DUP         x -- 0 | x x                DUP if nonzero
;Z  ?NEGATE      n1 n2 -- n3                 negate n1 if n2 negative
;Z  ?NUMBER      c-addr -- c-addr 0          if convert error
;Z  ?NUMBER      c-addr -- n -1              string->number
;Z  ?SIGN        adr n -- adr' n' f          get optional sign
;Z  ?branch      x --                        branch if TOS zero
;C  @            a-addr -- x                 fetch cell from memory
;C  ABORT        i*x -- R: j*x --            clear stk & QUIT
;C  ABORT"       i*x 0 -- i*x R: j*x -- j*x  x1=0
;C  ABORT"       i*x x1 -- R: j*x --         x1<>0
;C  ABS          n1 -- +n2                   absolute value
;C  ACCEPT       c-addr +n -- +n'            get line from term'l
;X  AGAIN        adrs --                     uncond'l backward branch
;C  ALIGN        --                          align HERE
;C  ALIGNED      addr -- a-addr              align given addr
;C  ALLOT        n --                        allocate n bytes in dict
;C  AND          x1 x2 -- x3                 logical AND
;Z  APP          -- a-addr                   xt of app ( was TURNKEY)
;U  APPCRC       -- crc                      CRC of APP-dictionary
;Z  APPU0        -- adr                      start of Application user area
;U  AT-XY        x y --                      send esc-sequence to terminal
;C  BASE         -- a-addr                   holds conversion radix
;C  BEGIN        -- adrs                     target for bwd. branch
;U  BELL         --                          send $07 to Terminal
;U  BIN          --                          set number base to binary
;C  BL           -- char                     an ASCII space
;Z  BOOT         --                          boot system
;C  C!           char c-addr --              store char in memory
;C  C,           char --                     append char to dict
;C  C@           c-addr -- char              fetch char from memory
;C  CAPITALIZE c-addr -- c-addrcapitalize string
;Z  CAPS         -- a-addr                   capitalize words
;U  CCLR         mask addr --                reset bit from mask in addr (byte)
;Z  CELL         -- n                        size of one cell
;C  CELL+        a-addr1 -- a-addr2          add cell size
;C  CELLS        n1 -- n2                    cells->adrs units
;U  CGET         mask addr -- flag           test bit from mask in addr (byte)
;C  CHAR         -- char                     parse ASCII character
;C  CHAR+        c-addr1 -- c-addr2          add char size
;C  CHARS        n1 -- n2                    chars->adrs units
;U  CLR          mask addr --                reset bit from mask in addr (cell)
;X  CMOVE        c-addr1 c-addr2 u --        move from bottom
;X  CMOVE>       c-addr1 c-addr2 u --        move from top
;Z  COLD         --                          set user area to latest application
;Z  COMPILE      --                          append inline execution token
;X  COMPILE,     xt --                       append execution token
;C  CONSTANT     --                          define a Forth constant
;Z  COR          -- adr                      cause of reset
;C  COUNT        c-addr1 -- c-addr2 u        counted->adr/len
;C  CR           --                          output newline
;C  CREATE       --                          create an empty definition
;U  CSET         mask addr --                set bit from mask in addr (byte)
;U  CTOGGLE      mask addr --                flip bit from mask in addr (byte)
;Z  D->I         c-addr1 c-addr2 u --        move Data->Code
;X  DABS         d1 -- +d2                   absolute value dbl.prec.
;C  DECIMAL      --                          set number base to decimal
;C  DEPTH        -- +n                       number of items on stack
;Z  DIGIT?       c -- n -1                   if c is a valid digit
;Z  DIGIT?       c -- x                      0
;X  DNEGATE      d1 -- d2                    negate double precision
;C  DO           -- adrs L: -- 0             start a loop
;C  DOES>        --                          change action of latest def'n
;C  DROP         x --                        drop top of stack
;X  DUMP         adr n --                    dump memory
;C  DUP          x -- x x                    duplicate top of stack
;C  ELSE         adrs1 -- adrs2              branch for IF..ELSE
;C  EMIT         c --                        output character to console
;Z  ENDLOOP      adrs xt -- L: 0 a1 a2 .. aN --common factor of LOOP and +LOOP
;C  ENVIRONMENT? c-addr u -- false           system query
;U  ESC[         --                          start esc-sequence
;C  EVALUATE     i*x c-addr u -- j*x         interprt string
;C  EXECUTE      i*x xt -- j*x               execute Forth word at 'xt'
;C  EXIT         --                          exit a colon definition
```

```
;Z  FACTORY      --                              set user area to delivery condition
;C  FILL         c-addr u char --                fill memory with char
;C  FIND         c-addr -- c-addr 0              if not found
;C  FIND         c-addr -- xt                    1
;C  FIND         c-addr -- xt -1                 if "normal"
;Z  FLALIGNED    a -- a'                         align IDP to flash boundary
;Z  FLERASE      a-addr n --                     erase n bytes of flash, full segment sizes.
;C  FM/MOD       d1 n1 -- n2 n3                  floored signed div'n
;U  GREEN        -- mask port                    green LED mask and port address
;Z  HEADER       --                              create a Forth word header
;C  HERE         -- addr                         returns dictionary ptr
;X  HEX          --                              set number base to hex
;Z  HIDE         --                              "hide" latest definition
;C  HOLD         char --                         add char to output string
;C  I            -- n R: sys1 sys2 -- sys1 sys2get the innermost loop index
;Z  I!           x a-addr --                     store cell in Instruction memory
;C  I,           x --                            append cell to Code dict
;Z  I->D         c-addr1 c-addr2 u --            move Code->Data
;Z  I@           a-addr -- x                     fetch cell from Instruction memory
;C  IALLOT       n --                            allocate n bytes in Code dict
;Z  IC!          x a-addr --                     store char in Instruction memory
;C  IC,          char --                         append char to Code dict
;Z  IC@          a-addr -- x                     fetch char from Instruction memory
;Z  ICOUNT       c-addr1 -- c-addr2 u            counted->adr/len
;Z  IDP          -- a-addr                       ROM dictionary pointer
;C  IF           -- adrs                         conditional forward branch
;C  IHERE        -- addr                         returns Code dictionary ptr
;Z  IMMED?       nfa -- f                        fetch immediate flag
;C  IMMEDIATE    --                              make last def'n immediate
;Z  INFOB        -- adr                          start of info B segment
;Z  INTERPRET    i*x c-addr u -- j*x             interpret given buffer
;C  INVERT       x1 -- x2                        bitwise inversion
;C  IS"          -- adr n                        compile in-line string
;Z  ITHERE       -- adr                          find first free flash cell
;Z  ITYPE        c-addr +n --                    type line to term'l
;Z  IWORD        c -- c-addr                     WORD to Code space
;Z  IWORDC       c -- c-addr                     maybe capitalize WORD to Code space
;C  J            -- n R: 4*sys -- 4*sys          get the second loop index
;C  KEY          -- c                            get character from keyboard
;X  KEY?         -- f                            return true if char waiting
;Z  L>           -- x L: x --                    move from leave stack
;C  LEAVE        -- L: -- adrs
;C  LITERAL      x --                            append numeric literal
;C  LOOP         adrs -- L: 0 a1 a2 .. aN --     finish a loop
;Z  LP           -- a-addr                       Leave-stack pointer
;C  LSHIFT       x1 u -- x2                      logical L shift u places
;C  M*           n1 n2 -- d                      signed 16*16->32 multiply
;X  M+           d n -- d                        add single to double
;X  MARKER       --                              create word to restore dictionary
;C  MAX          n1 n2 -- n3                     signed maximum
;U  MEM          -- u                            bytes left in flash
;Z  MEMBOT       -- adr                          begining of flash
;Z  MEMTOP       -- adr                          end of flash
;C  MIN          n1 n2 -- n3                     signed minimum
;C  MOD          n1 n2 -- n3                     signed remainder
;C  MOVE         addr1 addr2 u --                smart move
;U  MS           n --                            wait about n milliseconds
;Z  N=           c-addr1 c-addr2 u -- n          name compare
;Z  N=           n<0: s1<s2, n=0: s1=s2, n>0: s1>s2
;C  NEGATE       x1 -- x2                        two's complement
;Z  NEWEST       -- a-addr                       temporary LATEST storage
;Z  NFA>CFA      nfa -- cfa                      name adr -> code field
;Z  NFA>LFA      nfa -- lfa                      name adr -> link field
;X  NIP          x1 x2 -- x2                     per stack diagram
;Z  NOOP         --                              do nothing
;C  OR           x1 x2 -- x3                     logical OR
;C  OVER         x1 x2 -- x1 x2 x1               per stack diagram
;Z  P1           --                              adr
;Z  P2           --                              adr
;Z  P3           --                              adr
;X  PAD          -- a-addr                       user PAD buffer
;U  PAGE         --                              send "page" command to terminal to clear screen.
;U  PN           --                              send parameter of esc-sequence
;C  POSTPONE     --                              postpone compile action of word
;C  QUIT         -- R: i*x --                    interpret from kbd
;C  R>           -- x R: x --                    pop from return stack
```

```
;C  R@          -- x R: x -- x               fetch from rtn stk
;C  RECURSE     --                           recurse current definition
;U  RED         -- mask port                 red LED mask and port address
;C  REPEAT      adrs2 adrs1 --               resolve WHILE loop
;Z  REVEAL      --                           "reveal" latest definition
;C  ROT         x1 x2 x3 -- x2 x3 x1         per stack diagram
;Z  RP!         a-addr --                    set return stack pointer
;Z  RP@         -- a-addr                    get return stack pointer
;C  RSHIFT      x1 u -- x2                   logical R shift u places
;U  S2          -- mask port                 second button mask and port address
;U  S2?         -- f                         test button S2, true if pressed
;Z  S=          c-addr1 c-addr2 u -- n       string compare
;Z  S=          n<0: s1<s2, n=0: s1=s2, n>0: s1>s2
;C  S>D         n -- d                       single -> double prec.
;U  SAVE        --                           save user area to infoB
;Z  SCAN        c-addr u c -- c-addr' u'     find matching char
;U  SET         mask addr --                 set bit from mask in addr (cell)
;C  SIGN        n --                         add minus sign if n<0
;Z  SKIP        c-addr u c -- c-addr' u'     skip matching chars
;C  SM/REM      d1 n1 -- n2 n3               symmetric signed div
;C  SOURCE      -- adr n                     current input buffer
;Z  SP!         a-addr --                    set data stack pointer
;Z  SP@         -- a-addr                    get data stack pointer
;C  SPACE       --                           output a space
;C  SPACES      n --                         output n spaces
;C  STATE       -- a-addr                    holds compiler state
;C  SWAP        x1 x2 -- x2 x1               swap top two items
;C  THEN        adrs --                      resolve forward branch
;U  TOGGLE      mask addr --                 flip bit from mask in addr (cell)
;X  TUCK        x1 x2 -- x2 x1 x2            per stack diagram
;C  TYPE        c-addr +n --                 type line to term'l
;C  U.          u --                         display u unsigned
;X  U.R         u n --                       display u unsigned in n width
;C  U<          u1 u2 -- flag                test u1<u2, unsigned
;X  U>          u1 u2 -- flag                u1>u2 unsgd (not ANSI)
;Z  UD*         ud1 d2 -- ud3                32*16->32 multiply
;Z  UD/MOD      ud1 u2 -- u3 ud4             32/16->32 divide
;C  UM*         u1 u2 -- ud                  unsigned 16x16->32 mult.
;C  UM/MOD      ud u1 -- u2 u3               unsigned 32/16->16
;C  UNLOOP      -- R: sys1 sys2 --           drop loop parms
;C  UNTIL       adrs --                      conditional backward branch
;U  UNUSED      -- u                         bytes left in RAM
;C  UPC         char -- char                 capitalize character
;Z  USER        n --                         define user variable 'n'
;U  VALID?      -- f                         check if user app crc matches infoB
;C  VARIABLE    --                           define a Forth VARIABLE
;Z  WARM        --                           use user area from RAM (hopefully intact)
;C  WHILE       adrs1 -- adrs2 adrs1         branch for WHILE loop
;U  WIPE        --                           erase flash but not kernel, reset user area.
;X  WITHIN      n1|u1 n2|u2 n3|u3 -- f       n2<=n1<n3?
;C  WORD        char -- c-addr n             word delim'd by char
;X  WORDS       --                           list all words in dict.
;C  XOR         x1 x2 -- x3                  logical XOR
;X  ZERO        -- 0                         put zero on stack. Often usesd word.
;C  [           --                           enter interpretive state
;C  [']         --                           find word & compile as literal
;C  [CHAR]      --                           compile character literal
;U  \           --                           backslash
;C  ]           --                           enter compiling state
;Z  branch      --                           branch always
;U  ccrc        n c -- n'                    crc process byte
;U  crc         addr len -- n                crc process string
;Z  dp          -- a-addr                    holds dictionary ptr
;Z  hp          -- a-addr                    HOLD pointer
;Z  l0          -- a-addr                    bottom of Leave stack
;Z  latest      -- a-addr                    last word in dict.
;Z  lit         -- x                         fetch inline literal to stack
;Z  r0          -- a-addr                    end of return stack
;Z  s0          -- a-addr                    end of parameter stack
;X  tib         -- a-addr                    Terminal Input Buffer
;Z  tibsize     -- n                         size of TIB
;Z  u0          -- a-addr                    current user area adrs
;Z  uinit       -- addr                      initial values for user area
;Z  umax        u1 u2 -- u                   unsigned maximum
;Z  umin        u1 u2 -- u                   unsigned minimum
```