

Functional Groups, as defined in source code files.

4e-core430G2553.s43

;C EXECUTE	i*x xt -- j*x	execute Forth word at 'xt'
;Z lit	-- x	fetch inline literal to stack
;C EXIT	--	exit a colon definition
;C VARIABLE	--	define a Forth VARIABLE
;C CONSTANT	--	define a Forth constant
;Z USER	n --	define user variable 'n'
;C DUP	x -- x x	duplicate top of stack
;C ?DUP	x -- 0 x x	DUP if nonzero
;C DROP	x --	drop top of stack
;C SWAP	x1 x2 -- x2 x1	swap top two items
;C OVER	x1 x2 -- x1 x2 x1	per stack diagram
;C ROT	x1 x2 x3 -- x2 x3 x1	per stack diagram
;X NIP	x1 x2 -- x2	per stack diagram
;C >R	x -- R: -- x	push to return stack
;C R>	-- x R: x --	pop from return stack
;C R@	-- x R: x -- x	fetch from rtn stk
;Z SP@	-- a-addr	get data stack pointer
;Z SP!	a-addr --	set data stack pointer
;Z RP@	-- a-addr	get return stack pointer
;Z RP!	a-addr --	set return stack pointer
;X TUCK	x1 x2 -- x2 x1 x2	per stack diagram
;C @	a-addr -- x	fetch cell from memory
;C !	x a-addr --	store cell in memory
;C C@	c-addr -- char	fetch char from memory
;C C!	char c-addr --	store char in memory
;Z FLERASE	a-addr n --	erase n bytes of flash, full segment sizes.
;Z I!	x a-addr --	store cell in Instruction memory
;Z IC!	x a-addr --	store char in Instruction memory
;Z I@	a-addr -- x	fetch cell from Instruction memory
;Z IC@	a-addr -- x	fetch char from Instruction memory
;Z D->I	c-addr1 c-addr2 u --	move Data->Code
;C +	n1/u1 n2/u2 -- n3/u3	add n1+n2
;C +!	n/u a-addr --	add cell to memory
;X M+	d n -- d	add single to double
;C -	n1/u1 n2/u2 -- n3/u3	subtract n1-n2
;C AND	x1 x2 -- x3	logical AND
;C OR	x1 x2 -- x3	logical OR
;C XOR	x1 x2 -- x3	logical XOR
;C INVERT	x1 -- x2	bitwise inversion
;C NEGATE	x1 -- x2	two's complement
;C 1+	n1/u1 -- n2/u2	add 1 to TOS
;C 1-	n1/u1 -- n2/u2	subtract 1 from TOS
;Z ><	x1 -- x2	swap bytes (not ANSI)
;C 2*	x1 -- x2	arithmetic left shift
;C 2/	x1 -- x2	arithmetic right shift
;C LSHIFT	x1 u -- x2	logical L shift u places
;C RSHIFT	x1 u -- x2	logical R shift u places
;C 0=	n/u -- flag	return true if TOS=0
;C 0<	n -- flag	true if TOS negative
;C =	x1 x2 -- flag	test x1=x2
;X <>	x1 x2 -- flag	test not eq (not ANSI)
;C <	n1 n2 -- flag	test n1<n2, signed
;C >	n1 n2 -- flag	test n1>n2, signed
;C U<	u1 u2 -- flag	test u1<u2, unsigned
;X U>	u1 u2 -- flag	u1>u2 unsgd (not ANSI)
;Z branch	--	branch always
;Z ?branch	x --	branch if TOS zero
;Z (do)	n1 u1 n2 u2 -- R: -- sys1 sys2	run-time code for DO
;Z (loop)	R: sys1 sys2 -- sys1 sys2	run-time code for LOOP
;Z (+loop)	n -- R: sys1 sys2 -- sys1 sys2	run-time code for +LOOP
;C I	-- n R: sys1 sys2 -- sys1 sys2	get the innermost loop index
;C J	-- n R: 4*sys -- 4*sys	get the second loop index
;C UNLOOP	-- R: sys1 sys2 --	drop loop parms
;C UM*	u1 u2 -- ud	unsigned 16x16->32 mult.
;C UM/MOD	ud u1 -- u2 u3	unsigned 32/16->16

```

;C FILL          c-addr u char --          fill memory with char
;X CMOVE        c-addr1 c-addr2 u --      move from bottom
;X CMOVE>       c-addr1 c-addr2 u --      move from top
;Z I->D         c-addr1 c-addr2 u --      move Code->Data
;Z SKIP         c-addr u c -- c-addr' u'  skip matching chars
;Z SCAN        c-addr u c -- c-addr' u'  find matching char
;Z S=          c-addr1 c-addr2 u -- n      string compare
;Z S=          n<0: s1<s2, n=0: s1=s2, n>0: s1>s2
;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 EMIT        c --                      output character to console
;C KEY         -- c                      get character from keyboard
;X KEY?        -- f                      return true if char waiting
;X ZERO        -- 0                      put zero on stack. Often used word.

```

4e-deps430G2553.s43

```

;C ALIGN        --                      align HERE
;C ALIGNED     addr -- a-addr           align given addr
;Z CELL        -- n                    size of one cell
;C CELL+       a-addr1 -- a-addr2      add cell size
;C CELLS       n1 -- n2                cells->adrs units
;C CHAR+       c-addr1 -- c-addr2      add char size
;C CHARS       n1 -- n2                chars->adrs units
;C >BODY       xt -- a-addr            adrs of CREATE data
;X COMPILER,   xt --                   append execution token
;Z !CF         adrs cfa --              set code action of a word
;Z ,CF         adrs --                  append a code field
;Z ,CALL       adrs --                  append a subroutine CALL
;Z ,JMP        adrs --                  append an absolute 16-bit JMP
;Z !COLON      --                      change code field to DOCOLON
;Z ,EXIT       --                      append hi-level EXIT action
;Z ,BRANCH     xt --                   append a branch instruction
;Z ,DEST       dest --                  append a branch address
;Z !DEST       dest adrs --             change a branch dest'n
;Z ,NONE       --                      append a null destination (Flashable)

```

4e-hilvl430G2553.s43

```

; SYSTEM VARIABLES & CONSTANTS =====
;Z u0          -- a-addr                current user area adrs
;C >IN        -- a-addr                holds offset into TIB
;C BASE       -- a-addr                holds conversion radix
;C STATE      -- a-addr                holds compiler state
;Z dp         -- a-addr                holds dictionary ptr
;Z 'source    -- a-addr                two cells: len, adrs
;Z latest     -- a-addr                last word in dict.
;Z hp         -- a-addr                HOLD pointer
;Z LP         -- a-addr                Leave-stack pointer
;Z IDP        -- a-addr                ROM dictionary pointer
;Z NEWEST     -- a-addr                temporary LATEST storage
;Z APP        -- a-addr                xt of app ( was TURNKEY)
;Z CAPS       -- a-addr                capitalize words
;X PAD        -- a-addr                user PAD buffer
;Z l0         -- a-addr                bottom of Leave 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
;C BL         -- char                  an ASCII space
;Z uinit      -- addr                  initial values for user area
;Z #init      -- n                    #bytes of user area init data
;Z COR        -- adr                   cause of reset
;Z INFOB      -- adr                   start of info B segment
;Z APPU0      -- adr                   start of Application user area
; ARITHMETIC OPERATORS =====
;C S>D        n -- d                    single -> double prec.
;Z ?NEGATE    n1 n2 -- n3              negate n1 if n2 negative
;C ABS        n1 -- +n2                absolute value
;X DNEGATE    d1 -- d2                 negate double precision
;Z ?DNEGATE   d1 n -- d2              negate d1 if n negative
;X DABS       d1 -- +d2                absolute value dbl.prec.
;C M*         n1 n2 -- d               signed 16*16->32 multiply

```

```

;C SM/REM      d1 n1 -- n2 n3      symmetric signed div
;C FM/MOD      d1 n1 -- n2 n3      floored signed div'n
;C *           n1 n2 -- n3         signed multiply
;C /MOD        n1 n2 -- n3 n4      signed divide/rem'dr
;C /           n1 n2 -- n3         signed divide
;C MOD         n1 n2 -- n3         signed remainder
;C */MOD       n1 n2 n3 -- n4 n5   n1*n2/n3, rem'quot
;C */          n1 n2 n3 -- n4      n1*n2/n3
;C MAX         n1 n2 -- n3         signed maximum
;C MIN         n1 n2 -- n3         signed minimum
; DOUBLE OPERATORS =====
;C 2@          a-addr -- x1 x2     fetch 2 cells
;C 2!          x1 x2 a-addr --     store 2 cells
;C 2DROP       x1 x2 --            drop 2 cells
;C 2DUP        x1 x2 -- x1 x2 x1 x2 dup top 2 cells
;C 2SWAP       x1 x2 x3 x4 -- x3 x4 x1 x2 per diagram
;C 2OVER       x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2
; INPUT/OUTPUT =====
;C COUNT       c-addr1 -- c-addr2 u counted->adr/len
;C CR          --                  output newline
;C SPACE       --                  output a space
;C SPACES      n --                output n spaces
;Z umin        u1 u2 -- u          unsigned minimum
;Z umax        u1 u2 -- u          unsigned maximum
;C ACCEPT    c-addr +n -- +n'    get line from term'l
;C TYPE        c-addr +n --       type line to term'l
;Z ICOUNT      c-addr1 -- c-addr2 u counted->adr/len
;Z ITYPE       c-addr +n --       type line to term'l
;Z (IS")       -- c-addr u        run-time code for S"
;Z (S")        -- c-addr u        run-time code for S"
;C IS"         -- adr n           compile in-line string
;C ."         --                  compile string to print
;Z IWORD       c -- c-addr         WORD to Code space
;Z IWORDC      c -- c-addr         maybe capitalize WORD to Code space
; NUMERIC OUTPUT =====
;Z UD/MOD      ud1 u2 -- u3 ud4    32/16->32 divide
;Z UD*         ud1 d2 -- ud3       32*16->32 multiply
;C HOLD        char --            add char to output string
;C <#         --                  begin numeric conversion
;Z >digit      n -- c             convert to 0..9A..Z
;C #           ud1 -- ud2         convert 1 digit of output
;C #S          ud1 -- ud2         convert remaining digits
;C #>         ud1 -- c-addr u     end conv., get string
;C SIGN        n --              add minus sign if n<0
;C U.          u --              display u unsigned
;C .           n --              display n signed
;C DECIMAL     --                set number base to decimal
;X HEX         --                set number base to hex
; DICTIONARY MANAGEMENT =====
;C HERE        -- addr           returns dictionary ptr
;C ALLOT       n --              allocate n bytes in dict
;C ,           x --              append cell to dict
;C C,          char --           append char to dict
;C IHERE       -- addr           returns Code dictionary ptr
;C IALLOT      n --              allocate n bytes in Code dict
;C I,          x --              append cell to Code dict
;C IC,         char --           append char to Code dict
; INTERPRETER =====
;C SOURCE      -- adr n          current input buffer
;X /STRING     a u n -- a+n u-n   trim string
;Z >counted    src n dst --      copy to counted str
;C WORD        char -- c-addr n   word delim'd by char
;Z NFA>LFA     nfa -- lfa        name adr -> link field
;Z NFA>CFA     nfa -- cfa        name adr -> code field
;Z IMMED?      nfa -- f          fetch immediate flag
;C FIND        c-addr -- c-addr 0 if not found
;C FIND        c-addr -- xt      1
;C FIND        c-addr -- xt -1   if "normal"
;C UPC         char -- char       capitalize character
;C CAPITALIZE  c-addr -- c-addr capitalize string
;C LITERAL     x --              append numeric literal

```

```

;Z DIGIT?      c -- n -1          if c is a valid digit
;Z DIGIT?      c -- x          0
;Z ?SIGN       adr n -- adr' n' f  get optional sign
;C >NUMBER     ud adr u -- ud' adr' u'  convert string to number
;Z ?NUMBER     c-addr -- n -1      string->number
;Z ?NUMBER     c-addr -- c-addr 0    if convert error
;Z INTERPRET   i*x c-addr u -- j*x  interpret given buffer
;C EVALUATE    i*x c-addr u -- j*x  interprt string
;C QUIT        -- R: i*x --        interpret from kbd
;C ABORT       i*x -- R: j*x --     clear stk & QUIT
;Z ?ABORT      f c-addr u --       abort & print msg
;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 '           -- xt                find word in dictionary
;C CHAR        -- char              parse ASCII character
;C [CHAR]      --                   compile character literal
;C (           --                   skip input until )
; COMPILER =====
;Z HEADER      --                   create a Forth word header
;Z <BUILDS     --                   define a word with t.b.d. action & no data
;C CREATE      --                   create an empty definition
;Z (DOES>)    --                   run-time action of DOES>
;C DOES>      --                   change action of latest def'n
;C RECURSE    --                   recurse current definition
;C [          --                   enter interpretive state
;C ]          --                   enter compiling state
;Z HIDE        --                   "hide" latest definition
;Z REVEAL      --                   "reveal" latest definition
;C IMMEDIATE   --                   make last def'n immediate
;C :          --                   begin a colon definition
;C ;          --                   end a colon definition
;C [']        --                   find word & compile as literal
;C POSTPONE    --                   postpone compile action of word
;Z COMPILER    --                   append inline execution token
; CONTROL STRUCTURES =====
;C IF          -- adrs              conditional forward branch
;C THEN        adrs --              resolve forward branch
;C ELSE        adrs1 -- adrs2       branch for IF..ELSE
;C BEGIN       -- adrs              target for bwd. branch
;C UNTIL       adrs --              conditional backward branch
;X AGAIN       adrs --              uncond'l backward branch
;C WHILE       adrs1 -- adrs2 adrs1 branch for WHILE loop
;C REPEAT      adrs2 adrs1 --       resolve WHILE loop
;Z >L          x -- L: -- x          move to leave stack
;Z L>          -- x L: x --         move from leave stack
;C DO          -- adrs L: -- 0       start a loop
;Z ENLOOP      adrs xt -- L: 0 a1 a2 .. aN --common factor of LOOP and +LOOP
;C LOOP        adrs -- L: 0 a1 a2 .. aN -- finish a loop
;C +LOOP       adrs -- L: 0 a1 a2 .. aN -- finish a loop
;C LEAVE       -- L: -- adrs
; OTHER OPERATIONS =====
;X WITHIN      n1lu1 n2lu2 n3lu3 -- f  n2<=n1<n3?
;C MOVE        addr1 addr2 u --      smart move
;C DEPTH       -- +n                 number of items on stack
;C ENVIRONMENT? c-addr u -- false    system query
; UTILITY WORDS =====
;Z NOOP        --                   do nothing
;Z FLALIGNED   a -- a'               align IDP to flash boundary
;X MARKER      --                   create word to restore dictionary
;X WORDS       --                   list all words in dict.
;X U.R         u n --                 display u unsigned in n width
;X DUMP        adr n --               dump memory
;X .S          --                   print stack contents
;U ccrcc       n c -- n'             crc process byte
;U (crc        n addr len -- n'      crc process string including previous crc-byte
;U crc         addr len -- n         crc process string
; STARTUP WORDS =====
;Z ITHHERE     -- adr                find first free flash cell
;U APPCRC      -- crc                CRC of APP-dictionary
;U VALID?      -- f                  check if user app crc matches infoB
;U SAVE        --                   save user area to infoB

```

```

;Z BOOT      --          boot system
;Z WARM      --          use user area from RAM (hopefully intact)
;U .COLD     --          display COLD message
;Z COLD      --          set user area to latest application
;Z FACTORY   --          set user area to delivery condition
;U WIPE      --          erase flash but not kernel, reset user area.
;U MISC =====
;C ZCONSTANT --          define a Forth double constant
;U \         --          backslash
;Z .VER      --          type message
;U BELL      --          send $07 to Terminal
;U ESC[      --          start esc-sequence
;U PN        --          send parameter of esc-sequence
;U ;PN       --          send delimiter ; followed by parameter
;U AT-XY     x y --      send esc-sequence to terminal
;U PAGE      --          send "page" command to terminal to clear screen.
;U BIN       --          set number base to binary
;U MCU specific words =====
;U 1MS       --          wait about 1 millisecond
;U MS        n --      wait about n milliseconds
;U Bit manipulation words -----
;U CSET      mask addr -- set bit from mask in addr (byte)
;U CCLR      mask addr -- reset bit from mask in addr (byte)
;U CTOGGLE   mask addr -- flip bit from mask in addr (byte)
;U CGET      mask addr -- flag test bit from mask in addr (byte)
;U SET       mask addr -- set bit from mask in addr (cell)
;U CLR       mask addr -- reset bit from mask in addr (cell)
;U TOGGLE    mask addr -- flip bit from mask in addr (cell)
;U Memory info -----
;Z MEMBOT    -- adr      begining of flash
;Z MEMTOP    -- adr      end of flash
;U MEM       -- u        bytes left in flash
;U UNUSED    -- u        bytes left in RAM
;U MCU Peripherie -----
;Z P1        --          adr
;Z P2        --          adr
;Z P3        --          adr

```

4e-infoBG2553.s43

4e-init430G2553.s43

4e-LaunchPad.s43

```

;U MSP-EXP430G2 LaunchPad =====
;U PORT1 -----
;U RED       -- mask port red LED mask and port address
;U GREEN     -- mask port green LED mask and port address
;U S2        -- mask port second button mask and port address
;U S2?      -- f        test button S2, true if pressed

```

4e-vecs430G2553.s43