eForth Overview by C. H. Ting

Here just code. The complete file can be found at www.exemark.com under Forth

```
\ The following registers are required
\ for a virtual Forth computer:
\  Forth Register  8086 Register   Function
\  IP              SI              Interpreter Pointer
\  SP              SP              Data Stack Pointer
\  RP              RP              Return Stack Pointer
\  WP              AX              Word or Work Pointer
\  UP              (in memory )    User Area Pointer

Variable IP   \ Interpreter  Pointer
Variable SP   \ Data Stack   Pointer
Variable RP   \ Return Stack Pointer
Variable WP   \ Word or Work Pointer
Variable UP   \ User Area    Pointer

\ $NEXT    MACRO                   \ ------------------------------------1
           LODSW                   \ load next word into WP (AX)
           JMP     AX              \ jump directly to the word thru WP
           ENDM                    \ IP (SI) now points to the next word
\ doLIST   ( a -- )                \ Run address list in a colon word.          \ 2
           XCHG    BP,SP           \ exchange pointers
           PUSH    SI              \ push return stack
           XCHG    BP,SP           \ restore the pointers
           POP     SI              \ new list address
           $NEXT
\ CODE EXIT                        \ Terminate a colon definition.              \ 3
           XCHG    BP,SP           \ exchange pointers
           POP     SI              \ pop return stack
           XCHG    BP,SP           \ restore the pointers
           $NEXT
```

1.2 Memory Map - Memory used in eForth is separated into the following areas:

```
\ Cold boot       100H-17FH       Cold start and variable initial values
\ Code dictionary 180H-1344H      Code dictionary growing upward
\ Free space      1346H-33E4H     Shared by code/name dictionaries
\ Name/word       33E6H-3BFFH     Name dictionary growing downward
\ Data stack      3C00H-3E7FH     Growing downward
\ TIB             3E80H-          Growing upward
\ Return stack    -3F7FH                Growing downward
\ User variables  3F80H-3FFFH
```

;; Memory allocation

```
\ ;; 0//code>--//<<name//up>--<sp//tib>--rp//em
\ EM    EQU  04000H          ;top of memory
\ COLDD EQU  00100H          ;cold start vector
\ US    EQU  64*CELLL        ;user area size in cells
\ RTS   EQU  64*CELLL        ;return stack size
\ RPP   EQU  EM-8*CELLL      ;start of return stack (RP0)
\ TIBB  EQU  RPP-RTS         ;terminal input buffer (TIB)
\ SPP   EQU  TIBB-8*CELLL    ;start of data stack (SP0)
\ UPP   EQU  EM-256*CELLL    ;start of user area (UP0)
\ NAMEE EQU  UPP-8*CELLL     ;name dictionary
\ CODEE EQU  COLDD+US        ;code dictionary
```

2.0 eForth Kernel

```
\ System interface:    BYE,   ?rx,    tx!,   !io
\ Inner interpreters:  doLIT, doLIST, next,  ?branch,  branch, EXECUTE,  EXIT
\ Memory access:       !  ,   @,     C!,    C@
\ Return stack:        RP@,   RP!,    R@,    R>
\ Data stack:          SP@,   SP!,    DROP,  DUP,    SWAP,   OVER
\ Logic:               0<,    AND,    OR,    XOR
\ Arithmetic:          UM+
```

2.1 System Interface

```
\   CODE BYE( -- , exit Forth )          \ ----------------------------- 4
           INT     020H                 \ return to DOS
\
\ CODE  ?RX  ( -- c T | F )             \ Return input character and true,     \ --5
\                                       \ or a false if no input.
\            $CODE   3,'?RX',QRX
\            XOR     BX,BX              \ BX=0 setup for false flag
\            MOV     DL,0FFH            \ input command
\            MOV     AH,6               \ MS-DOS Direct Console I/O
\            INT     021H
\            JZ      QRX3               \ ?key ready
\            OR      AL,AL              \ AL=0 if extended char
\            JNZ     QRX1               \ ?extended character code
\            INT     021H
\            MOV     BH,AL              \ extended code in msb
\            JMP     QRX2
\ QRX1:      MOV     BL,AL
\ QRX2:      PUSH    BX                 \ save character
\            MOV     BX,-1              \ true flag
\ QRX3:      PUSH    BX
\            $NEXT
\
\ CODE  TX!  ( c -- )                   \ Send character c to output device. \ --- 6
\            POP     DX                 \ char in DL
\            CMP     DL,0FFH            \ 0FFH is interpreted as input
\            JNZ     TX1                \ do NOT allow input
\            MOV     DL,32              \ change to blank
\ TX1:       MOV     AH,6               \ MS-DOS Direct Console I/O
\            INT     021H               \ display character
\            $NEXT
\
\ CODE  !IO  ( -- )                     \ Initialize the serial I/O devices.  \ --7
\          $NEXT
```

2.2 Inner Interpreter

```
\ $NEXT   MACRO                         \ ------------------------------------ 8
\         LODSW                         \ load next word into WP (AX)
\         JMP     AX                    \ jump directly to the word thru WP
\         ENDM                          \ IP (SI) now points to the next word
\
\ doLIST  ( a -- )                      \ Run address list in a colon word.     \ 9
\         XCHG    BP,SP                 \ exchange pointers
\         PUSH    SI                    \ push return stack
\         XCHG    BP,SP                 \ restore the pointers
\         POP     SI                    \ new list address
\         $NEXT
\
\ CODE  EXIT                            \ Terminate a colon definition.     \ -- 10
\         XCHG    BP,SP                 \ exchange pointers
\         POP     SI                    \ pop return stack
\         XCHG    BP,SP                 \ restore the pointers
\         $NEXT
\
\ CODE  EXECUTE  ( ca -- )              \ Execute the word at ca.           \ -- 11
\         POP     BX
\         JMP     BX                    \ jump to the code address
\
\ CODE  doLIT  ( -- w )                 \ Push inline literal on data stack.   \ 12
\         LODSW                         \ get the literal compiled in-line
\         PUSH    AX                    \ push literal on the stack
\         $NEXT                         \ execute next word after literal
```

2.3 Loops and Branches

```
\ CODE  next ( -- )                     \ Decrement index and exit loop      \ - 13
\         SUB     WORD PTR [BP],1       \ if index is less than 0.
\         JC      NEXT1                 \ decrement the index
\         MOV     SI,0[SI]              \ ?decrement below 0
\         $NEXT                         \ no, continue loop
\ NEXT1:  ADD     BP,2                  \ yes, pop the index
\         ADD     SI,2                  \ exit loop
\         $NEXT
\
\ CODE  ?branch  ( f -- )               \ Branch if flag is zero.           \---14
\         POP     BX                    \ pop flag
\         OR      BX,BX                 \ ?flag=0
\         JZ      BRAN1                 \ yes, so branch
\         ADD     SI,2                  \ point IP to next cell
\         $NEXT
\ BRAN1:  MOV     SI,0[SI]              \ IP:=(IP), jump to new address
\         $NEXT
\
\ CODE  branch  ( -- )                  \ Branch to an inline address.      \ --15
\         MOV     SI,0[SI]              \ jump to new address unconditionally
\         $NEXT
```

2.4 Memory Access

```
\ CODE  !        ( w a -- )     \ Pop the data stack to memory.            \ --16
\         POP     BX             \ get address from tos
\         POP     0[BX]          \ store data to that adddress
\         $NEXT
\
\ CODE  @        ( a -- w ) \ Push memory location to data stack.           \ 17
\         POP     BX             \ get address
\         PUSH    0[BX]          \ fetch data
\         $NEXT
\
\ CODE  C!   ( c b -- )    \ Pop data stack to byte memory.                  \ --18
\         POP     BX             \ get address
\         POP     AX             \ get data in a cell
\         MOV     0[BX],AL       \ store one byte
\         $NEXT
\
\ CODE  C@  ( b -- c )     \ Push byte memory content on data stack.          \ 19
\         POP     BX             \ get address
\         XOR     AX,AX          \ AX=0 zero the hi byte
\         MOV     AL,0[BX]       \ get low byte
\         PUSH    AX             \ push on stack
\         $NEXT
```

2.5 Return Stack

```
\ CODE  RP@      ( -- a )  \ Push current RP to data stack.                \ -- 20
\         PUSH    BP             \ copy address to return stack
\         $NEXT                  \ pointer register BP
\
\ CODE  RP!      ( a -- ) \ Set the return stack pointer.                  \ ---- 21
\         POP     BP             \ copy (BP) to tos
\         $NEXT
\
\ CODE  R>       ( -- w )   \ Pop return stack to data stack.               \ -- 22
\         PUSH    0[BP]          \ copy w to data stack
\         ADD     BP,2           \ adjust RP for popping
\         $NEXT
\
\ CODE  R@       ( -- w )   \ Copy top of return stack to data stack.       \ 23
\         PUSH    0[BP]          \ copy w to data stack
\         $NEXT
\
\ CODE  >R       ( w -- )   \ Push data stack to return stack.              \ -- 24
\         SUB     BP,2           \ adjust RP for pushing
\         POP     0[BP]          \ push w to return stack
\         $NEXT
```

2.6 Data Stack

```
\ CODE  DROP     ( w -- ) \ Discard top stack item.                \ ---------- 25
\         ADD     SP,2           \ adjust SP to pop
\         $NEXT
\
\ CODE  DUP      ( w -- w w )  \ Duplicate the top stack item.              \ 26
\         MOV     BX,SP          \ use BX to index the stack
\         PUSH    0[BX]
\         $NEXT
\
\ CODE  SWAP  ( w1 w2 -- w2 w1 ) \ Exchange top two stack items.            \ 27
\         POP BX                 \ get w2
\         POP AX                 \ get w1
\         PUSH            BX             \ push w2
\         PUSH            AX             \ push w1
\         $NEXT
\
\ CODE  OVER  ( w1 w2 -- w1 w2 w1 ) \ Copy second stack item to top         \ 28
\         MOV     BX,SP          \ use BX to index the stack
\         PUSH    2[BX]          \ get w1 and push on stack
\         $NEXT
\
\ CODE  SP@      ( -- a )  \ Push the current data stack pointer.          \ 29
\         MOV     BX,SP          \ use BX to index the stack
\         PUSH    BX             \ push SP back
\         $NEXT
\
\ CODE  SP!      ( a -- )  \ Set the data stack pointer.                   \ 30
\         POP             SP             \ safety
\         $NEXT
```

2.7 Logical Words

```
\ CODE  0<       ( n -- f )    \ Return true if n is negative.             \ --- 31
\         POP     AX
\         CWD                    \ sign extend AX into DX
\         PUSH    DX             \ push 0 or -1
\         $NEXT
\
\ CODE  AND      ( w w -- w )  \ Bitwise AND.                              \ ---- 32
\         POP     BX
\         POP     AX
\         AND     BX,AX
\         PUSH    BX
\         $NEXT
\
\ CODE  OR       ( w w -- w )  \ Bitwise inclusive OR.                     \  33
\         POP     BX
\         POP     AX
\         OR      BX,AX
\         PUSH    BX
\         $NEXT
\
\ CODE  XOR      ( w w -- w )  \ Bitwise exclusive OR.                     \  34
\         POP     BX
\         POP     AX
\         XOR     BX,AX
\         PUSH    BX
\         $NEXT
```

2.8 Primitive Arithmetic

```
\ CODE  UM+   ( w w -- w cy )   \                          ------------------------------ 35
\                                   \ Add two numbers,
\                                   \ return the sum and carry flag.
\         XOR     CX,CX    \ CX=0 initial carry flag
\         POP     BX
\         POP     AX
\         ADD     AX,BX
\         RCL     CX,1           \ get carry
\         PUSH    AX                   \ push sum
\         PUSH    CX                        \ push carry
\         $NEXT
```

3.0 High Level Forth Words

3.1 Variables and User Variables

```
\ : doVAR ( -- a ) R> ; \ starting at 100 rather than continue              \ 101
\
\ VARIABLE UP  ( -- a, Pointer to the user area.)                     \ ----- - 102
\
\ : doUSER  ( -- a, Run time routine for user variables.)                  \ -- 103
\           R> @ ;                      \ retrieve user area offset
\           UP @ + ;                    \ add to user area base addr
\
\ : doVOC  ( -- ) R> CONTEXT ! ;                     \ --------------------- 104
\ : FORTH  ( -- ) doVOC [ 0 , 0 ,                    \ --------------------- 105
\ : doUSER ( -- a ) R> @ UP @ + ;                    \ --------------------- 106
\
\ SP0      ( -- a, pointer to bottom of the data stack.)                   \ 107
\ RP0      ( -- a, pointer to bottom of the return stack.)                 \ 108
\ '?KEY    ( -- a, execution vector of ?KEY. Default to ?rx.)              \ 109
\ 'EMIT    ( -- a, execution vector of EMIT.  \ Default to tx!)            \ 110
\
\ 'EXPECT  ( -- a, execution vector of EXPECT.  Default to 'accept'.)      \ 111
\
\ 'TAP     ( -- a, execution vector of TAP.  Defulat the kTAP.)            \ 112
\
\ 'ECHO    ( -- a, execution vector of ECHO.  Default to tx!.)             \ 113
\
\ 'PROMPT  ( -- a, execution vector of PROMPT.  Default to '.ok'.)         \ 115
\
\ BASE     ( -- a,.radix base for numeric I/O.  \ Default to 10.)          \ 116
\
\ tmp      ( -- a, a temporary storage location used in parse and find.)   \ 117
\
\ SPAN     ( -- a, hold character count received by EXPECT.)               \ 118
\ >IN      ( -- a, hold the character pointer                              \ 119
\                  while parsing input stream.)
\
\ #TIB     ( -- a, hold the current count and address                      \ 120
\                  of the terminal input buffer.
\                  Terminal Input Buffer used one cell after #TIB.)
\
\ CSP      ( -- a, hold the stack pointer for error checking.)             \ 121
\ 'EVAL    ( -- a, execution vector of EVAL. Default to EVAL.)             \ 122
\ 'NUMBER  ( -- a, address of number conversion.                           \ 123
\                  Default to NUMBER?.)
\ HLD      ( -- a, hold a pointer in building a numeric output string.)    \ 124
\
\ HANDLER  ( -- a, hold the return stack pointer for error                 \ 125
\                  handling.)
\ CONTEXT  ( -- a, a area to specify vocabulary search order               \ 126
\                  Default to FORTH.
\                  Vocabulary stack, 8 cells follwing CONTEXT.)
\ CURRENT  ( -- a, point to the vocabulary to be extended.                 \ 127
\                  Default to FORTH.
\                  Vocabulary link uses one cell after CURRENT.)
\ CP       ( -- a, point to the top of the code dictionary.)               \ 128
\ NP       ( -- a, point to the bottom of the name dictionary.)            \ 129
\ LAST     ( -- a, point to the last name in the name dictionary.)
\                                                                          \ 130
```

3.2 Common Functions

```
\  ?DUP,  ROT,   2DROP, and  2DUP  \  are stack operators supplementing
\                                  \  the four classic stack operators
\  DUP,   SWAP,  OVER   and  DROP.
\
\ : ?DUP     ( w -- w w | 0 )       DUP IF DUP THEN ;                      \ 131
\ : ROT      ( w1 w2 w3 -- w2 w3 w1 ) >R SWAP R> SWAP ;                    \ 132
\ : 2DROP    ( w w -- )             DROP DROP ;                            \ 133
\ : 2DUP     ( w1 w2 -- w1 w2 w1 w2 ) OVER OVER ;                          \ 134
\ : +        ( w w -- w )           UM+ DROP ;                             \ 135
\ : NOT      ( w -- w )             -1 XOR ;                               \ 136
\ : NEGATE   ( n -- -n )            NOT 1 + ;                              \ 137
\ : DNEGATE  ( d -- -d )            NOT >R NOT 1 UM+ R> + ;                \ 138
\ : D+       ( d d -- d )           >R SWAP >R UM+ R> R> + + ;             \ 139
\ : -        ( w w -- w )           NEGATE + ;                             \ 140
\ : ABS      ( n -- +n )            DUP 0< IF NEGATE THEN ;                \ 141
```

3.3 Comparison

```
\ : =        ( w w -- f )      XOR IF 0 EXIT THEN -1 ;                     \ 142
\ : U<       ( u u -- t )      2DUP XOR 0< IF SWAP DROP 0< EXIT THEN - 0< ; \ 143
\ : <        ( n n -- t )      2DUP XOR 0< IF       DROP 0< EXIT THEN - 0< ; \ 144
\ : MAX      ( n n -- n )      2DUP        < IF SWAP THEN DROP ;           \ 145
\ : MIN      ( n n -- n )      2DUP SWAP < IF SWAP THEN DROP ;             \ 146
\ : WITHIN   ( u ul uh -- t )  OVER - >R - R> U< ;   \ ul <= u < uh        \ 147
\
```

3.4 Divide

```
\ : UM/MOD  ( ud u -- ur uq )                                              \ 148
\   2DUP U<
\   IF NEGATE  15
\     FOR >R DUP UM+ >R >R DUP UM+ R> + DUP
\       R> R@ SWAP >R UM+  R> OR
\       IF >R DROP 1 + R> ELSE DROP THEN R>
\     NEXT DROP SWAP EXIT
\   THEN DROP 2DROP  -1 DUP ;
\
\ : M/MOD   ( d n -- r q ) \ floored division                              \ 149
\   DUP 0<  DUP >R
\   IF NEGATE >R DNEGATE R>
\   THEN >R DUP 0< IF R@ + THEN R> UM/MOD R>
\   IF SWAP NEGATE SWAP THEN ;
\
\ : /MOD    ( n n -- r q ) OVER 0< SWAP M/MOD ;                            \ 150
\ : MOD     ( n n -- r )  /MOD DROP ;                                      \ 151
\ : /       ( n n -- q )  /MOD SWAP DROP ;                                 \ 152
```

3.5 Multiply

```
\ : UM*     ( u u -- ud )                                                  \ 153
\   0 SWAP  ( ul 0 u2 ) 15
\   FOR DUP UM+ >R >R DUP UM+ R> + R>
\     IF >R OVER UM+ R> + THEN
\   NEXT ROT DROP ;
\ : *       ( n n -- n ) UM* DROP ;                                        \ 154
\ : M*      ( n n -- d )                                                   \ 155
\   2DUP XOR 0< >R.  ABS SWAP ABS UM*  R> IF DNEGATE THEN ;
\ : */MOD   ( n n n -- r q ) >R M* R> M/MOD ;                              \ 156
\ : */      ( n n n -- q ) */MOD SWAP DROP ;                               \ 157
```

3.6 Memory Alignment

```
\ : CELL-   ( a -- a ) -2 + ;                                              \ 158
\ : CELL+   ( a -- a ) 2 + ;                                               \ 159
\ : CELLS   ( n -- n ) 2 * ;                                               \ 160
\ : ALIGNED ( b -- a )                                                     \ 161
\     DUP 0 2 UM/MOD DROP DUP
\       IF 2 SWAP - THEN + ;
\ : BL      ( -- 32 ) 32 ;                                                 \ 162
\ : >CHAR   ( c -- c )                                                     \ 163
\     $7F AND DUP 127 BL WITHIN IF DROP 95 THEN ;
\ : DEPTH   ( -- n ) SP@ SP0 @ SWAP - 2 / ;                                \ 164
\ : PICK    ( +n -- w ) 1 + CELLS SP@ + @ ;                                \ 165
```

3.7 Memory Access

```
\ : +!      ( n a -- ) SWAP OVER @ + SWAP ! ;                              \ 166
\ : 2!      ( d a -- ) SWAP OVER ! CELL+ ! ;                               \ 167
\ : 2@      ( a -- d ) DUP CELL+ @ SWAP @ ;                                \ 168
\ : COUNT   ( b -- b +n ) DUP 1 + SWAP C@ ;                                \ 169
\ : HERE    ( -- a ) CP @ ;                                                \ 170
\ : PAD     ( -- a ) HERE 80 + ;                                           \ 171
\ : TIB     ( -- a ) #TIB CELL+ @ ;                                        \ 172
\ : @EXECUTE ( a -- ) @ ?DUP IF EXECUTE THEN ;                             \ 173
\ : CMOVE   ( b b u -- )                                                   \ 174
\     FOR AFT >R DUP C@ R@ C! 1 + R> 1 + THEN NEXT 2DROP ;
\ : FILL    ( b u c -- )                                                   \ 175
\     SWAP FOR SWAP AFT 2DUP C! 1 + THEN NEXT 2DROP ;
\ : -TRAILING ( b u -- b u )                                               \ 176
\     FOR AFT BL OVER R@ + C@ <
\       IF R> 1 + EXIT THEN THEN
\     NEXT 0 ;
\ : PACK$   ( b u a -- a ) \ null fill                                     \ 177
\     ALIGNED  DUP >R OVER
\     DUP 0 2 UM/MOD DROP
\     - OVER +  0 SWAP !  2DUP C!  1 + SWAP CMOVE  R> ;
```

4.0 Text Interpreter

```
\     Accept text input from a terminal
\     Parse out commands from input text
\     Search dictionary
\     Execute commands
\     Translate numbers into binary
\     Display numbers in text form
\     Handle errors gracefully
```

4.1 Numeric Output

```
\ : DIGIT   ( u -- c )          9 OVER < 7 AND + 48 + ;                    \ 178
\ : EXTRACT ( n base -- n c ) 0 SWAP UM/MOD SWAP DIGIT ;                   \ 179
\ : <#      ( -- )              PAD HLD ! ;                                \ 180
\ : HOLD    ( c -- )            HLD @ 1 - DUP HLD ! C! ;                   \ 181
\ : #       ( u -- u )          BASE @ EXTRACT HOLD ;                      \ 182
\ : #S      ( u -- 0 )          BEGIN # DUP WHILE REPEAT ;                 \ 183
\ : SIGN    ( n -- )            0< IF 45 HOLD THEN ;                       \ 184
\ : #>      ( w -- b u )        DROP HLD @ PAD OVER - ;                    \ 185
\ : str     ( n -- b u )        DUP >R ABS <# #S R> SIGN #> ;              \ 186
\ : HEX     ( -- )              16 BASE ! ;                                \ 187
\ : DECIMAL ( -- )              10 BASE ! ;                                \ 188
```

4.2 Number Output

```
\ : str     ( n -- b u )                                                   \ 189
\     ( Convert a signed integer to a numeric string.)
\   DUP >R    ( save a copy for sign)
\   ABS       ( use absolute of n)
\   <# #S     ( convert all digits)
\   R> SIGN   ( add sign from n)
\   #> ;      ( return number string addr and length)
\ : HEX     ( -- )                                                         \ 190
\     ( Use radix 16 as base for numeric conversions.)
\   16 BASE ! ;
\ : DECIMAL ( -- )                                                         \ 191
\     ( Use radix 10 as base for numeric conversions.)
\   10 BASE ! ;
\ : .R      ( n +n -- )  \ Display an integer in a field                   \ 192
\                            of n columns, right justified.)
\     >R str      ( convert n to a number string)
\     R> OVER - SPACES ( print leading spaces)
\     TYPE ;  ( print number in +n column format)
\ : U.R     ( u +n -- )                                                    \ 193
\                 ( Display an unsigned integer in n column,
\                   right justified.)
\   >R        ( save column number)
\   <# #S #> R> ( convert unsigned number)
\   OVER - SPACES ( print leading spaces)
\   TYPE ;  ( print number in +n columns)
\ : U.      ( u -- )                                                       \ 194
\                 ( Display an unsigned integer in free format.)
\   <# #S #>  ( convert unsigned number)
\   SPACE     ( print one leading space)
\   TYPE ;    ( print number)
\ : .       ( w -- )                                                       \ 195
\                 ( Display an integer in free format,
\                   preceeded by a space.)
\   BASE @ 10 XOR   ( if not in decimal mode)
\     IF U. EXIT THEN ( print unsigned number)
\   str SPACE TYPE ;( print signed number if decimal)
\ : ?       ( a -- )                                                       \ 196
\                 ( Display the contents in a memory cell.)
\   @ . ;   ( very simple but useful command)
```

4.3 Numeric Input

```
\ : DIGIT?      ( c base -- u t )                                          \ 197
\   >R 48 - 9 OVER <
\     IF 7 - DUP 10 < OR THEN DUP R> U< ;
\ : NUMBER?     ( a -- n T | a F )                                         \ 198
\   BASE @ >R  0 OVER COUNT ( a 0 b n)
\   OVER C@ 36 =
\   IF HEX SWAP 1 + SWAP 1 - THEN ( a 0 b' n')
\   OVER C@ 45 = >R ( a 0 b n)
\   SWAP R@ - SWAP R@ + ( a 0 b" n") ?DUP
\   IF 1 - ( a 0 b n)
\     FOR DUP >R C@ BASE @ DIGIT?
\       WHILE SWAP BASE @ * +  R> 1 +
\     NEXT DROP R@ ( b ?sign) IF NEGATE THEN SWAP
\       ELSE R> R> ( b index) 2DROP ( digit number) 2DROP 0
\       THEN DUP
\   THEN R> ( n ?sign) 2DROP R> BASE ! ;
```

4.4 Basic I/O

```
\   : xxx  ...   " A compiled string"  ...   ;                             \ 199
\   : yyy  ...   ." An output string"  ...   ;                             \ 200
\
\ : ?KEY   ( -- c T  | F ) '?KEY @EXECUTE ;                               \ 201
\ : KEY    ( -- c )   BEGIN ?KEY UNTIL ;                                   \ 202
\ : EMIT   ( c -- )   'EMIT @EXECUTE ;                                     \ 203
\ : NUF?   ( -- f )   ?KEY DUP IF 2DROP KEY 13 = THEN ;                    \ 204
\ : PACE   ( -- )   11  EMIT ;                                             \ 205
\ : SPACE  ( -- )   BL  EMIT ;                                             \ 206
\ : CHARS  ( +n c -- ) \ ???ANS conflict                                   \ 207
\     SWAP 0 MAX FOR AFT DUP EMIT THEN NEXT DROP ;                         \ 208
\ : SPACES ( +n -- )   BL CHARS ;                                          \ 209
\
\ : TYPE   ( b u -- ) FOR AFT DUP C@ EMIT 1 + THEN NEXT DROP ;
\                                                                          \ 210
\ : CR     ( -- )     13 EMIT 10 EMIT ;                                    \ 211
\ : do$    ( -- a )                                                        \ 212
\   R> R@ R> COUNT + ALIGNED >R SWAP >R ;                                  \ 213
\ : $"|    ( -- a )   do$ ;                                               \ 214
\ : ."|    ( -- )     do$ COUNT TYPE ; COMPILE-ONLY                        \ 215
\ :  .R    ( n +n -- ) >R str        R> OVER - SPACES TYPE ;                \ 216
\ : U.R    ( u +n -- ) >R <# #S #> R> OVER - SPACES TYPE ;                 \ 217
\ : U.     ( u -- )    <# #S #> SPACE TYPE ;                               \ 218
\ :  .     ( n -- )    BASE @ 10 XOR IF U. EXIT THEN str SPACE TYPE ;      \ 219
\ : ?      ( a -- )  @ . ;                                                 \ 220
\
```

4.5 Parsing

```
\ : parse  ( b u c -- b u delta ; <string> )                               \ 221
\   tmp !  OVER >R  DUP \ b u u
\   IF 1 -  tmp @ BL =
\     IF \ b u' \ 'skip'
\       FOR BL OVER C@ - 0< NOT  WHILE 1 +
\       NEXT ( b) R> DROP 0 DUP EXIT \ all delim
\         THEN  R>
\     THEN OVER SWAP \ b' b' u' \ 'scan'
\     FOR tmp @ OVER C@ -  tmp @ BL =
\       IF 0< THEN WHILE 1 +
\     NEXT DUP >R  ELSE R> DROP DUP 1 + >R
\                  THEN OVER -  R>  R> - EXIT
\   THEN      ( b u) OVER R> - ;
\ : PARSE    ( c -- b u ; <string> )                                       \ 222
\   >R  TIB >IN @ +  #TIB @ >IN @ -  R> parse >IN +! ;
\ : .(       ( -- ) 41 PARSE TYPE ; IMMEDIATE                              \ 223
\ : (        ( -- )  41 PARSE 2DROP ; IMMEDIATE                            \ 224
\ : \ ( -- ) #TIB @ >IN ! ; IMMEDIATE                                      \ 225
\ : CHAR     ( -- c ) BL PARSE DROP C@ ;                                   \ 226
\ : TOKEN    ( -- a ; <string> )                                           \ 227
\   BL PARSE 31 MIN NP @ OVER - CELL- PACK$ ;                              \ 228
\ : WORD     ( c -- a ; <string> ) PARSE HERE PACK$ ;                      \ 229
```

4.6 Dictionary Search

```
\ : NAME>     ( a -- xt ) CELL- CELL- @ ;                                  \ 230
\ : SAME?     ( a a u -- a a f \ -0+ )                                     \ 231
\   FOR AFT OVER R@ CELLS + @
\                 OVER R@ CELLS + @ -   ?DUP
\     IF R> DROP EXIT THEN THEN
\   NEXT 0 ;
\ : find      ( a va -- xt na | a F )                                      \ 232
\   SWAP            \ va a
\   DUP C@ 2 / tmp !  \ va a   \ get cell count
\   DUP @ >R        \ va a   \ count byte & 1st char
\   CELL+ SWAP      \ a' va
\   BEGIN @ DUP     \ a' na na
\     IF DUP @ [ =MASK ] LITERAL AND  R@ XOR \ ignore lexicon bits
\       IF CELL+ -1 ELSE CELL+ tmp @ SAME? THEN
\     ELSE R> DROP EXIT
\     THEN
\   WHILE CELL- CELL- \ a' la
\   REPEAT R> DROP SWAP DROP CELL-  DUP NAME> SWAP ;
\ : NAME?     ( a -- xt na | a F )                                         \ 233
\   CONTEXT  DUP 2@ XOR IF CELL- THEN >R \ context<>also
\   BEGIN R>  CELL+  DUP >R  @  ?DUP
\   WHILE find  ?DUP
\   UNTIL R> DROP EXIT THEN R> DROP  0 ;
```

4.7 Terminal

```
\ : ^H        ( b b b -- b b b )      \ backspace                          \ 234
\   >R OVER R> SWAP OVER XOR
\   IF  8 'ECHO @EXECUTE
\       32 'ECHO @EXECUTE        \ distructive
\        8 'ECHO @EXECUTE        \ backspace
\   THEN ;
\ : TAP       ( bot eot cur c -- bot eot cur )                             \ 235
\   DUP 'ECHO @EXECUTE OVER C! 1 + ;
\ : kTAP      ( bot eot cur c -- bot eot cur )                             \ 236
\   DUP 13 XOR
\   IF 8 XOR IF BL TAP ELSE ^H THEN EXIT
\   THEN DROP SWAP DROP DUP ;
\ : accept    ( b u -- b u )                                               \ 237
\   OVER + OVER
\   BEGIN 2DUP XOR
\   WHILE  KEY  DUP BL -  95 U<
\     IF TAP ELSE 'TAP @EXECUTE THEN
\   REPEAT DROP  OVER - ;
\ : EXPECT    ( b u -- ) 'EXPECT @EXECUTE SPAN ! DROP ;                    \ 238
\ : QUERY     ( -- )                                                       \ 239
\   TIB 80 'EXPECT @EXECUTE #TIB !  DROP 0 >IN ! ;
```

4.8 Error Handling

```
\ : CATCH          ( ca -- err#/0 )                                        \ 240
\                  ( Execute word at ca and set up an error frame for it.)
\   SP@ >R         ( save current stack pointer on return stack )
\   HANDLER @ >R    ( save the handler pointer on return stack )
\   RP@ HANDLER !   ( save the handler frame pointer in HANDLER )
\   ( ca ) EXECUTE  ( execute the assigned word
\                        over this safety net )
\ R> HANDLER !     ( normal return from the executed word )
\                  ( restore HANDLER from the return stack )
\ R> DROP          ( discard the saved data stack pointer )
\ 0 ;              ( push a no-error flag on data stack )
\ : THROW      ( err# -- err# )                                            \ 241
\     ( Reset system to current local error frame an update error flag.)
\     HANDLER @ RP! ( expose latest error handler frame
\                     on return stack )
\     R> HANDLER !   ( restore previously saved error handler frame )
\     R> SWAP >R     ( retrieve the data stack pointer saved )
\     SP!            ( restore the data stack )
\     DROP
\     R>  ;          ( retrived err# )
\ : CATCH      ( xt -- 0 | err# )                                          \ 242
\     SP@ >R  HANDLER @ >R  RP@ HANDLER !
\     EXECUTE
\     R> HANDLER !  R> DROP  0 ;
\ : THROW ( err# -- err# )                                                 \ 243
\     HANDLER @ RP!  R> HANDLER !  R> SWAP >R SP! DROP R> ;
\
\     CREATE NULL$ 0 , $," coyote"                                         \ 244
\ : ABORT      ( -- ) NULL$ THROW ;                                        \ 245
\ : abort"     ( f -- ) IF do$ THROW THEN do$ DROP ;                       \ 246
\     BEGIN QUERY [ ' EVAL ] LITERAL   CATCH                               \ 247
\     ?DUP UNTIL                                                           \ 248
\ : ABORT   NULL$ THROW ;                                                  \ 249
\ : abort"   IF do$ THROW THEN do$ DROP ;                                  \ 250
\ : ?STACK   DEPTH 0< IF $" underflow" THROW THEN ;                        \ 251
\ : $INTERPRET  ... 'NUMBER @EXECUTE  IF EXIT THEN THROW ;                \ 251a
```

4.9 Text Interpreter

```
\ : $INTERPRET   ( a -- )                                                  \ 252
\     NAME?  ?DUP
\       IF @ $40 AND
\     ABORT" compile ONLY" EXECUTE EXIT
\       THEN 'NUMBER @EXECUTE IF EXIT THEN THROW ;
\ : [ ( -- ) doLIT $INTERPRET 'EVAL ! ; IMMEDIATE                          \ 253
\ : .OK ( -- ) doLIT $INTERPRET 'EVAL @ = IF ." ok" THEN CR ;              \ 254
\ : ?STACK ( -- ) DEPTH 0< ABORT" underflow" ;                             \ 255
\ : EVAL ( -- )                                                            \ 256
\   BEGIN TOKEN DUP C@
\   WHILE 'EVAL @EXECUTE ?STACK
\   REPEAT DROP 'PROMPT @EXECUTE ;
```

4.10 Shell

```
\ : PRESET   ( -- ) SP0 @ SP!  TIB #TIB CELL+ ! ;                          \ 257
\ : xio      ( a a a -- ) \ reset  'EXPECT 'TAP  'ECHO 'PROMPT             \ 258
\     doLIT accept  'EXPECT 2! 'ECHO 2! ; COMPILE-ONLY
\ : FILE ( -- )                                                            \ 259
\     doLIT PACE  doLIT DROP  doLIT kTAP xio ;
\ : HAND ( -- )                                                            \ 260
\     doLIT .OK   doLIT EMIT  [ kTAP  xio ;
\ CREATE I/O  ' ?RX , ' TX! , \ defaults                                   \ 262
\ : CONSOLE ( -- ) I/O 2@ '?KEY 2! HAND ;                                  \ 263
\ : QUIT ( -- )                                                            \ 264
\     RP0 @ RP!
\     BEGIN [COMPILE] [
\       BEGIN QUERY doLIT EVAL CATCH ?DUP
\       UNTIL 'PROMPT @ SWAP CONSOLE  NULL$ OVER XOR
\       IF CR #TIB 2@ TYPE
\          CR >IN @ 94 CHARS
\          CR COUNT TYPE ."  ? "
\       THEN doLIT .OK XOR
\       IF $1B EMIT THEN
\     PRESET
\     AGAIN ;
```

5.0 eForth Compiler

5.1 Interpreter and Compiler

```
\ : [       ( -- )                                                         \ 265
\     [ ' $INTERPRET ] LITERAL
\     'EVAL !         ( vector EVAL to $INTERPRET )
\   ; IMMEDIATE       ( enter into text interpreter mode )
\ : ] ( -- )                                                               \ 266
\     [ ' $COMPILE ] LITERAL
\     'EVAL !         ( vector EVAL to $COMPILE )
\ ;
```

5.2 Primitive Compiler Words

```
\ : '          ( -- xt )          TOKEN NAME? IF EXIT THEN THROW ;         \ 267
\ : ALLOT      ( n -- )           CP +! ;                                  \ 268
\ : ,          ( w -- )           HERE DUP CELL+ CP ! ! ; \ ???ALIGNED     \ 269
\ : [COMPILE]  ( -- ; <string> )  ' , ; IMMEDIATE                          \ 270
\ : COMPILE    ( -- )             R> DUP @ , CELL+ >R ;                    \ 271
\ : LITERAL    ( w -- )           COMPILE doLIT , ; IMMEDIATE              \ 272
\ : $,"        ( -- )             34 WORD COUNT ALIGNED CP ! ;             \ 273
\ : RECURSE    ( -- )             LAST @ NAME> , ; IMMEDIATE               \ 274
```

5.3 Structures

```
\ Conditional branch        IF    ... THEN
\                           IF    ... ELSE ...  THEN
\ Finite loop               FOR   ... NEXT
\                           FOR   ... AFT   ... THEN ... NEXT
\ Infinite loop             BEGIN ... AGAIN
\ Indefinite loop           BEGIN ... UNTIL
\                           BEGIN ... WHILE ... REPEAT
\ : <MARK    ( -- a )       HERE ;                                         \ 275
\ : <RESOLVE ( a -- )     , ;                                             \ 276
\ : >MARK    ( -- A )       HERE 0 , ;                                     \ 277
\ : >RESOLVE ( A -- )       <MARK SWAP ! ;                                 \ 278
\ : FOR      ( -- a )       COMPILE >R <MARK ; IMMEDIATE                   \ 279
\ : BEGIN    ( -- a )       <MARK ; IMMEDIATE                              \ 280
\ : NEXT     ( a -- )       COMPILE next <RESOLVE ; IMMEDIATE              \ 281
\ : UNTIL    ( a -- )       COMPILE ?branch <RESOLVE ; IMMEDIATE           \ 282
\ : AGAIN    ( a -- )       COMPILE  branch <RESOLVE ; IMMEDIATE           \ 283
\ : IF       ( -- A )       COMPILE ?branch >MARK ; IMMEDIATE              \ 284
\ : AHEAD    ( -- A )       COMPILE branch >MARK ; IMMEDIATE               \ 285
\ : REPEAT   ( A a -- )     [COMPILE] AGAIN >RESOLVE ; IMMEDIATE           \ 286
\ : THEN     ( A -- )       >RESO
```