CHAPTER 4. THE FORTH KERNAL

4.1. MEMORY AND I/O PORT WORDS

The power of Forth over other high level languages is that it allows the use to manipulate directly the addresses and the contents of memory. It gives the user the freedom of assembly language and the ease of usage in BASIC. However, the user must bear the responsibility of this freedom. If you stores something into the wrong place, you can easily crash the system.

LaForth includes words to access memory outside of the 64K byte code-data segment. These extended memory words are prefixed by X; i.e., X@, XC@, etc.

@ (addr -- n)

Replace address on stack by its value. Use top as an address to get word which replaces it.

```
          DB        0
          DB        "@"
          CHAIN     0
AT:       POP       BX
          PUSH      [BX]
          NEXT
```

C@ (addr -- n)

Replace the address at the top of the stack with the contents of the byte at the specified address. 8 high zero bits are appended to the byte.

```
          DB        0
          DB        "@C"
          CHAIN     C
CAT:      POP       BX
          MOV       AL,[BX]
          XOR       AH,AH
          PUSH      AX
          NEXT
```

2@ (addr -- d)

Replace the address on the top of the stack with the double number found at the address.

```
          HEADER    !@2,R
TWOAT:    POP       BX
          PUSH      [BX+2]
          PUSH      [BX]
          NEXT
```

X@ (seg addr -- n)

Replace the address and segment with the contents of the word addressed.

```
          HEADER    !@X,X
XAT:      MOV       DX,ES
          POP       BX
          POP       ES
          MOV       AX,ES:[BX]
          MOV       ES,DX
          PUSH      AX
          NEXT
```

XC@ (seg addr -- n)

Replace the address and segment with the contents of the byte addressed.

```
          HEADER    !@CX,X
```

```
XCAT:     MOV       DX,ES
          POP       BX
          POP       ES
          MOV       AL,ES:[BX]
          XOR       AH,AH
          PUSH      AX
          MOV       ES,DX
          NEXT
```

P@ (addr -- val)

Fetch the 16 bit value from the port address specified. Replace port address with word contents of port.

```
          HEADER    !@P,P
PAT:      POP       DX
          IN        AX,DX
          PUSH      AX
          NEXT
```

PC@ (addr -- val)

Fetch the 8 bit value from the port address specified. Replace port address with byte contents of port.

```
          HEADER    !@CP,P
PCAT:     POP       DX
          IN        AL,DX
          XOR       AH,AH
          PUSH      AX
          NEXT
```

! (n addr --)

Store the second item at the address on top.

```
          HEADER    !!,A
STORE:    POP       BX
STO1:     POP       AX
          MOV       [BX],AX
          NEXT
```

2! (d addr --)

Store the 32 bit value at the memory address specified.

```
          HEADER    !!2,R
TWOSTO:   POP       BX
          POP       AX
          MOV       [BX],AX
          ADD       BX,2
          JMP       STO1
```

C! (n addr --)

Store the least significant 8 bits of the value 2nd on the stack at the address specified on the top of the stack.

```
          HEADER    !!C,C
CSTOR:    POP       BX
          POP       AX
          MOV       [BX],AL
          NEXT
```

+! (n addr --)

Add the second item to the value addressed on top.

```
          HEADER    !!+,K
PLSTOR:   POP       BX
          POP       AX
          ADD       [BX],AX
          NEXT
```

X!   (val seg addr --)

Store the 16 bit value which is 3rd on the stack at the address specified by the segment and offset. Drop all three items from ;the stack.

```
            HEADER    !!X,X
XSTOR:      MOV       DX,ES
            POP       BX
            POP       ES
            POP       AX
            MOV       ES:[BX],AX
            MOV       ES,DX
            NEXT
```

XC!   (val seg addr --)

Store the 8 bit value which is 3rd on the stack at the address specified by the segment and offset. Drop all three items from ;the stack.

```
            HEADER    !!CX,X
XCSTOR:     MOV       DX,ES
            POP       BX
            POP       ES
            POP       AX
            MOV       ES:[BX],AL
            MOV       ES,DX
            NEXT
```

P!   (val addr --)

Store the 16 bit value at the port address specified.

```
            HEADER    !!P,P
PSTOR:      POP       DX
            POP       AX
            OUT       DX,AX
            NEXT
```

PC!   (val addr --)

Store the 8 bit value at the port address specified.

```
            HEADER    !!CP,P
PCSTOR:     POP       DX
            POP       AX
            OUT       DX,AL
            NEXT
```

CMOVE   (from to count --)

Smart character move. Top=byte count, 2nd=to address, 3rd=from address. Moves one byte at a time from low addresses to higher addresses. This is a "smart" move, which does not cause a fill.

```
            HEADER    EVOMC,C
CMOVE:      POP       CX                    ; Count of bytes
            POP       BX                    ; To-address
            POP       AX                    ; From-address
            PUSH      SI                    ; Save SI
            PUSH      DI                    ; and DI
            PUSH      ES                    ; and ES.
            MOV       DX,CS                 ; Make ES the same as CS.
            MOV       ES,DX
            MOV       SI,AX                 ; From-address
            MOV       DI,BX                 ; To-address
            CMP       SI,DI                 ; See if we should reverse the move
            JNC       CM1
            ADD       SI,CX                 ; Yes.  Start at end of the string
```

```
          ADD       DI,CX
          SUB       SI,2
          STD                           ; Reverse the Direction flag
CM1:      REP       MOVSB               ; Do the move
          CLD                           ; Restore the Direction flag
          POP       ES                  ; Restore ES,
          POP       DI                  ; DI,
          POP       SI                  ; and SI.
          NEXT
```

FILL (addr cnt val --)
Fills a block of memory with any specific character. Top=the fill character, 2nd is the count of characters, 3rd is the low order address of the block.

```
          HEADER    LLIF,F
FILL:     POP       AX                  ; Value to fill with.
          POP       CX                  ; Byte count.
          POP       BX                  ; Destination address.
          PUSH      DI                  ; Save some registers
          PUSH      ES
          MOV       DX,CS
          MOV       ES,DX
          MOV       DI,BX
          OR        CX,CX               ; Check for zero count case.
          JZ        FILLX
          REP       STOSB               ; Repeat the Store String 'cnt' times.
FILLX:    POP       ES                  ; Restore registers.
          POP       DI
          NEXT                          ; and exit.
```

4.2. RETURN STACK WORDS

The return stack in LaForth is placed at the top of the extra memory segment pointed to by ES:DI register pair. This 64K byte segment of memory is immediately above the code-data-stack segment pointed to by CS, DS and SS. The lower portion of this extra segment is used as a text buffer to read source files from DOS.

The return stack is used to stack IP, addresses of words to be executed. It is also used to store values from the data stack temporarily. The third important usage is to store the loop counts of the FOR-NEXT type of loops. A FOR-NEXT loop holds a decrementing index on the return stack. When the loop index is decremented below 0, the loop is terminated. It is a much simpler structure than the traditional Forth DO-LOOP, which stores two or more items on the return stack.

+R (n --)
Add top to the value on top of R-Stack.

```
          HEADER    R+,K
RADD:     POP       AX
          ADD       ES:[DI]-2,AX
          NEXT
```

RDROP
Drop the top of the R-stack.

```
          HEADER    PORDR,R
RDROP:    SUB       DI,2
          NEXT
```

RP!

Restores the Return Stack to initial state. Clears the R-stack.

```
                HEADER      !!PR,R
RCLR:           MOV         DI,-256
                MOV         AX,CS
                ADD         AX,1000h
                MOV         ES,AX
                NEXT
```

>R (n --)
Moves top of stack to Return stack.

```
                DB          0
                DB          "R>"
                CHAIN       1E
TOR:            POP         AX
                STOSW
                NEXT
```

R> (-- n)
Moves top of Return stack to data stack.

```
                DB          0
                DB          ">R"
                CHAIN       R
FROMR:          SUB         DI,2
                PUSH        ES:[DI]
                NEXT
```

I (-- n)
Calculate and push the innermost loop index to the data stack.

```
                HEADER      I,I
I:              PUSH        ES:[DI-2]
                NEXT
```

2I (-- n)
Push 2* inner loop index. This is useful to get an address offset from the loop index.

```
                HEADER      I2,R
TWOI:           MOV         AX,ES:[DI-2]
                SHL         AX,1
                PUSH        AX
                NEXT
```

J (-- n)
Push second innermost loop index to the data stack.

```
                HEADER      J,J
                PUSH        ES:[DI-4]
                NEXT
```

4.3. DATA STACK WORDS

This set of data stack words includes all the stack words in most standard Forth system, with some additions like 2DUP, and @OVER, etc., to manipulate double integers, and other conveniences like NIP, and SWAB, which swaps bytes in the integer on the top of data stack.

SP@ (-- n)
Push current stack pointer. Gets the address of the top of the computational stack then uses that value.

```
                HEADER      !@PS,S
```

```
SPAT:       MOV         AX,SP                       ; Can't just PUSH SP .
            PUSH        AX
            NEXT

SP!         Clear parameter stack to its initial state.
            HEADER      !!PS,S
SPSTO:      MOV         SP,TOES
            NEXT

DUP         ( n -- n n )
Duplicate top of the stack.
            HEADER      PUD,D
XDUP:       POP         AX
            PUSH        AX
            PUSH        AX
            NEXT

?DUP        ( n -- n n ) or ( n -- n )
Duplicate top if non-zero.
            HEADER      PUD?,1F
QDUP:       MOV         BP,SP
            MOV         AX,[BP]
            OR          AX,AX
            JE          QDUP1
            PUSH        AX
QDUP1:      NEXT

OVER        ( n1 n2 -- n1 n2 n1 )
Copies second number to the top.
            HEADER      REVO,O
OVER:       MOV         BP,SP
            PUSH        [BP+2]
            NEXT

2DUP        ( d -- d d )
Duplicates top double number.
            HEADER      PUD2,R
TWODUP:     MOV         BP,SP
            PUSH        [BP+2]
            PUSH        [BP]
            NEXT

2OVER       ( d1 d2 -- d1 d2 d1 )
Duplicates top double number.
            HEADER      REVO2,R
TWOOVR:
            MOV         BP,SP
            PUSH        [BP+6]
            PUSH        [BP+4]
            NEXT

ROT         ( n3 n2 n1 -- n2 n1 n3 )
Rotate the top three items on stack.
            HEADER      TOR,R
ROT:        POP         AX
            POP         BX
            POP         CX
ROT1:       PUSH        BX
```

```
                PUSH        AX
                PUSH        CX
                NEXT
```

-ROT (n3 n2 n1 -- n1 n3 n2)
Rotate the top three items backwards.

```
                HEADER      TOR-,M
MROT:           POP         BX
                POP         CX
                POP         AX
                JMP         ROT1
```

DROP (n --)
Deletes the top number from the stack.

```
                HEADER      PORD,D
DROP:           ADD         SP,2
                NEXT
```

2DROP (n1 n2 --)
Deletes two words from the stack.

```
                HEADER      PORD2,R
DROP2:          ADD         SP,4
                NEXT
```

NIP (n1 n2 -- n2)
Deletes second word on stack.

```
                HEADER      PIN,N
NIP:            POP         AX
                ADD         SP,2
                PUSH        AX
                NEXT
```

SWAP (n1 n2 -- n2 n1)
Exchange top two values on the stack.

```
                HEADER      PAWS,S
SWAP:           POP         AX
                POP         BX
                PUSH        AX
                PUSH        BX
                NEXT
```

2SWAP (d1 d2 -- d2 d1)
Exchange the top two doubles on stack.

```
                HEADER      PAWS2,R
TWOSWP:         POP         BX
                POP         DX
                POP         CX
                POP         AX
                PUSH        DX
                JMP         ROT1
```

SWAB (n1 -- n2)
Swap bytes in the 16 bit number on the top of the stack.

```
                HEADER      BAWS,S
SWAB:           POP         AX
                XCHG        AL,AH
                PUSH        AX
                NEXT
```

4.4. COMPARISON WORDS

0= (n -- f)

Replace top by -1 if top is zero.

```
                HEADER          !=0,P
ZEQU:           POP             AX
ZEQU1:          OR              AX,AX
                JNE             SETZ
                JMP             SETM1
```

D0= (d -- f)

Set -1 on stack if top double is zero.

```
                HEADER          !=0D,D
DZEQU:          POP             BX
                POP             AX
                OR              BX,BX
                JNE             SETZ
                JMP             ZEQU1
```

0<> (n -- f)

Replace top with -1 if it is non-zero.

```
                DB              0,"><0"
                CHAIN           P
ZNE:            POP             AX
ZNE1:           OR              AX,AX
                JNE             SETM1
                PUSH            AX
                NEXT
```

0< (n -- f)

Replace top with -1 if it is negative.

```
                DB              0
                DB              "<0"
                CHAIN           P
ZLESS:          POP             AX
                OR              AX,AX
                JS              SETM1
                JMP             SETZ
```

< (n1 n2 -- f)

Replace top 2 with -1 if n1 < n2.

```
                DB              0
                DB              "<"
                CHAIN           1C
LESS:           POP             AX
                POP             BX
                CMP             BX,AX
                JL              SETM1
SETZ:           XOR             AX,AX
                PUSH            AX
                NEXT
```

D< (d1 d2 -- f)

Double less-than function. Set -1 if d1<d2.

```
            HEADER      !<D,D
DLESS:      POP         CX
            POP         AX
            POP         DX
            POP         BX
            CMP         DX,CX
            JL          SETM1
            JG          SETZ
            JMP         ULESS1
```

U< (n1 n2 -- f)

Unsigned Less-than function.

```
            DB          0
            DB          "<U"
            CHAIN       U
ULESS:      POP         AX
            POP         BX
ULESS1:     CMP         BX,AX
            JAE         SETZ
SETM1:      MOV         AX,-1
            PUSH        AX
            NEXT
```

UD< (d1 d2 -- f)

Unsigned Double Less-than function.

```
            HEADER      !<DU,U
UDLESS:     POP         CX
            POP         AX
            POP         DX
            POP         BX
            CMP         DX,CX
            JB          SETM1
            JNE         SETZ
            JMP         ULESS1
```

4.5. SIMPLE NUMBERS

RAND (-- n)

Pushes a word of random bits onto the stack. The seed is two words at INPTR-2 and INPTR-4. The sum of the two words in the seed must be odd.

```
            HEADER      DNAR,R
RAND1:      MOV         BX,RAND+2
            MOV         AX,RAND
            ADD         AX,RAND+2
            MOV         RAND+2,AX
            MOV         RAND,BX
            PUSH        AX
            NEXT
```

0 (-- 0)

```
            HEADER      0,P
ZERO:       XOR         AX,AX
            PUSH        AX
            NEXT
```

```
1           ( -- 1 )
            HEADER    1,Q
ONE:        MOV       AX,1
            PUSH      AX
            NEXT

2           ( -- 2 )
            HEADER    2,R
TWO:        MOV       AX,2
            PUSH      AX
            NEXT

3           ( -- 3 )
            HEADER    3,S
THREE:      MOV       AX,3
            PUSH      AX
            NEXT

4           ( -- 4 )
            HEADER    4,T
FOUR:       MOV       AX,4
            PUSH      AX
            NEXT

 -1         ( -- -1 )
            HEADER    1-,M
            MOV       AX,-1
            PUSH      AX
            NEXT

-2          ( -- -2 )
            HEADER    2-,M
            MOV       AX,-2
            PUSH      AX
            NEXT
```

4.6. ARITHMETIC WORDS

```
1+          ( n1 -- n2 )
Add 1 to top of stack.
            HEADER    +1,Q
ONEP:       POP       AX
            ADD       AX,1
            PUSH      AX
            NEXT

2+          ( n1 -- n2 )
Add 2 to top of stack.
            HEADER    +2,R
TWOP:       POP       AX
            ADD       AX,2
            PUSH      AX
            NEXT
```

```
1-          ( n1 -- n2 )
Subtract 1 from Top of stack.
            HEADER    -1,Q
ONEM:       POP       AX
            SUB       AX,1
            PUSH      AX
            NEXT

2-          ( n1 -- n2 )
Subtract 2 from TOS.
            HEADER    -2,R
TWOM:       POP       AX
            SUB       AX,2
            PUSH      AX
            NEXT

+           ( n1 n2 -- n3 )
 Replace top two items with the sum.
            HEADER    +,K
PLUS:       POP       AX
            POP       BX
            ADD       AX,BX
            PUSH      AX
            NEXT

D+          ( d1 d2 -- d3 )
Sum top two double numbers.
            HEADER    +D,D
DPLUS:      POP       DX
            POP       CX
            POP       BX
            POP       AX
            ADD       BX,DX
            ADC       AX,CX
            PUSH      BX
            PUSH      AX
            NEXT

-           ( n1 n2 -- n3 )
Replace top 2 with second minus top.
            HEADER    -,M
SUB:        POP       AX
            POP       BX
            SUB       BX,AX
            PUSH      BX
            NEXT

D-          ( d1 d2 -- d1-d2 )
Subtract top double from second double.
            HEADER    -D,D
DSUB:       POP       DX
            POP       CX
            POP       BX
            POP       AX
            SUB       BX,DX
            SBB       AX,CX
            PUSH      BX
            PUSH      AX
```

```
                NEXT
```

NEG (n1 -- n2)

Negate top word.

```
                HEADER      GEN,N
XNEG:           POP         AX
                NEG         AX
                PUSH        AX
                NEXT
```

DNEG (d1 -- d2)

Negate double word at top.

```
                HEADER      GEND,D
DNEG:           POP         AX
                POP         BX
                XOR         CX,CX
                NEG         BX
                SBB         CX,AX
                PUSH        BX
                PUSH        CX
                NEXT
```

M* (u1 u2 -- ud)

16-bit Multiply, 32-bit result Unsigned multiply 2nd by top. Leave 32 bit result with high order part on top and low order part 2nd.

```
                HEADER      *M,M
MMULT:          POP         AX                  ; Get the two arguments from the stack
                POP         BX
                MUL         BX                  ; Let MUL do the hard work
                PUSH        AX                  ; Then return the result to the stack
                PUSH        DX
                NEXT
```

UD* (ud1 ud2 -- uquad)

Unsigned multiply of two double precision numbers, yielding a quadruple precision result.

```
                HEADER      *DU,U
UDSTAR:         MOV         BP,SP               ; A B  C D
                MOV         AX,[BP+6]           ; D
                MOV         CX,[BP+2]           ; B
                MOV         BX,AX               ;
                MUL         CX                  ; D*B
                MOV         [BP+6],AX           ; DBL -> 4th on stack
                XCHG        DX,BX
                MOV         AX,[BP]             ; A
                MUL         DX                  ; D*A
                ADD         AX,BX               ; DAL+DBH
                ADC         DX,0
                MOV         BX,DX               ; DAH
                XCHG        AX,CX
                MOV         DX,[BP+4]           ; C
                MUL         DX                  ; C*B
                ADD         CX,AX               ; CBL+DAL+DBH
                ADC         BX,DX               ; DAH+CBH+CY1+CY2
                MOV         AX,[BP]             ; A
                MOV         DX,[BP+4]           ; C
                MOV         [BP+4],CX           ; CDL+DAL+DBH -> 3rd on stack
                MUL         DX                  ; A*C
                ADD         AX,BX               ; CAL+DAH+CBH
```

```
            ADC     DX,0
            MOV     [BP+2],AX                   ; CAL+DAH+CBH -> 2nd on stack
            MOV     [BP],DX                     ; CAH -> TOS
            NEXT
```

*   (n1 n2 -- n3)
Multiply top two stack elements. Multiply top by 2nd, leaving single precision signed product on top.

```
            HEADER  *,J
XMULT:      POP     AX
            POP     BX
            MUL     BX
            PUSH    AX
            NEXT
```

/MOD   (n1 n2 -- rem quot)
Treat the top two operands as unsigned numbers. Divide the second by the top. The quotient replaces the top value, and the remainder replaces the second value.

```
            HEADER  DOM/,O
SLMOD:      POP     CX
            XOR     DX,DX
            JMP     MDIV1
```

UM/MOD   (ud un -- rem quot)
Unsigned division with quotient and remainder.

```
            HEADER  DOM/MU,U
UMDIV:      POP     CX                          ; Divisor
            POP     DX                          ; MS part of numerator
MDIV1:      POP     AX                          ; LS part of numerator
            DIV     CX
            PUSH    DX                          ; Remainder
            PUSH    AX                          ; Quotient
            NEXT
```

UDMOD/   (uquad uddiv -- udquot udrem)
Divide a quad precision number by a double precision number. Note that the unsigned double quotient is 2nd on the stack and the unsigned double remainder is on the top.

```
            HEADER  /DOMDU,U
UDDIV:      POP     CX                          ; DenomHi
            POP     DX                          ; DenomLo
            POP     AX                          ; AccHi
            POP     BX                          ; AccLo
            MOV     BP,32                       ; Set count to 32
            PUSH    BP                          ; Keep the count on the stack.
            MOV     BP,SP                       ; Point to stack
            CLC                                 ; Not really needed
UD1:        RCL     WORD PTR [BP+4],1           ; Shift the 64 bit Accumulator left by 1
            RCL     WORD PTR [BP+2],1
            RCL     BX,1
            RCL     AX,1
            JNC     UD2                         ; If no carry, we must do a test subtraction.
UD1SUB:     SUB     BX,DX                       ; Carry was set: We must subtract.
            SBB     AX,CX                       ; AX is the most significant part.
            DEC     BYTE PTR [BP]               ; Decrement the counter
            STC                                 ;
            JNZ     UD1                         ; Continue until counter is zero
            JMP     UD3                         ; Go to trailer when nearly done.
UD2:        CMP     AX,CX                       ; Start comparison at MS word
            JC      UD2CC                       ; If carry is set, don't subtract.
```

```
        JNZ     UD1SUB                  ; If result is non-zero, do subtract.
        CMP     BX,DX                   ; Otherwise compare LS word
        JNC     UD1SUB                  ; If carry is clear, subtract.
UD2CC:  DEC     WORD PTR [BP]           ; Decrement the counter.
        CLC                             ; Clear the carry bit.
        JNZ     UD1                     ; Continue process till count is zero.

UD3:    RCL     WORD PTR 4[BP],1        ; Final adjustment of quotient.
        RCL     WORD PTR [BP+2],1
        MOV     [BP],BX                 ; Put LS of remainder on stack.
        PUSH    AX                      ; Push MS of remainder on stack.
        NEXT                            ; Normal ending.
```

2*   (n1 -- n2)
Signed left shift.

```
        HEADER  *2,R
MTWO:   POP     AX
        SHL     AX,1
        PUSH    AX
        NEXT
```

2/   (n1 -- n2)
Signed right shift.

```
        HEADER  /2,R
DTWO:   POP     AX
        SHR     AX,1
        PUSH    AX
        NEXT
```

D2*   (d1 -- d2)
Shift left by 1 the double number at the top of the stack.

```
        HEADER  *2D,D
DMTWO:  POP     AX
        POP     BX
        SHL     BX,1
        RCL     AX,1
        PUSH    BX
        PUSH    AX
        NEXT
```

D2/   (d1 -- d2)
Shift right by 1 the double number at the top of the stack. The most significant bit of the result is 0.

```
        HEADER  /2D,D
DDTWO:  POP     AX
        POP     BX
        SHR     AX,1
        RCR     BX,1
        PUSH    BX
        PUSH    AX
        NEXT
```

4.7. LOGIC WORDS

```
AND         ( n1 n2 -- n3 )
Logical "AND".
            HEADER      DNA,A
XAND:       POP         AX
            POP         BX
            AND         AX,BX
            PUSH        AX
            NEXT
```

```
XOR         ( n1 n2 -- n3 )
Exclusive OR.
            HEADER      ROX,X
XXOR:       POP         AX
            POP         BX
            XOR         AX,BX
            PUSH        AX
            NEXT
```

```
OR          ( n1 n2 -- n3 )
Inclusive OR.
      HEADER  RO,O
ORX:        POP         AX
            POP         BX
            OR          AX,BX
            PUSH        AX
            NEXT
```

```
COMP        ( n1 -- n2 )
Complement the 16 bit top word.
            HEADER      PMOC,C
XCOMP:      POP         AX
            XOR         AX,-1
            PUSH        AX
            NEXT
```

```
SCOMP       ( addr -- )
Complement bit # 7.  Complements the sign bit of the byte addressed by top.
            HEADER      PMOCS,S
SCOMP:      POP         BX
            MOV         AL,80h
            XOR         BYTE PTR [BX],AL
            NEXT
```