CHAPTER 5. TEXT INTERPRETER

5.1. CHARACTER INPUT

GETCHAR (-- char)
Obtain the next character from either the keyboard, or the current input buffer, whichever is active.
IF IN is 0, use input from the keyboard; otherwise, IN points to text in the text buffer. For input from the keyboard, bit 8 indicates an ALT or other special function key. Keyboard characters are normally echoed to the screen. This is the only way by which the text interpreter obtains the source characters. LaForth thus treats source code in files identically to that from the keyboard.

```
            HEADER  RAHCTEG,G
GTCHR:      CALL    NUCHAR
            AND     AX,01FFh
            PUSH    AX
            NEXT
```

Gets a character from either the keyboard or memory. Echos LF after a CR, but ignores the first LF after CR from input.

```
NUCHAR:
            MOV     BX,INPTR        ; Use keyboard input if INPTR is zero.
            OR      BX,BX
            JNE     BUFGET          ; Get from buffer
NUCH1:      CALL    EGET            ; Get from keyboard or equivalent
            XOR     BX,BX
            OR      AH,AH
            JNS     CRMOD
            CMP     AL,CRCH         ; Check for Carriage Return
            JNE     NUCH2
            MOV     AL,LFCH         ; Now send a Line-Feed
            CALL    QCOUT
            MOV     AL,CRCH         ; Restore the CR
            XOR     AH,AH
            MOV     BH,0FFh         ; Set a flag
            JMP     CRMOD
NUCH2:      OR      AH,AH
            JZ      CRMOD           ; If not a special character, return
            CMP     AL,LFCH         ; Check for Line-Feed
            JNE     NUCH4
            MOV     AH,CRSEEN       ; Check the CR flag
            OR      AH,AH
            JZ      NUCH3
            XOR     AH,AH           ; Clear flag
            MOV     CRSEEN,AH
            JMP     NUCH1           ; and ignore the Line-Feed
NUCH3:      CALL    QCOUT           ; Echo other Line-Feeds
            JMP     CRCLR
NUCH4:      CMP     AL,ESCCH        ; Escape Code
            JNE     CRCLR
            CALL    EGET            ; ESC key seen. Get next character,
CRCLR:      XOR     BH,BH           ; clear BH
CRMOD:      MOV     CRSEEN,BH       ; and CRSEEN flag.
            RET

BUFGET:     MOV     AL,ES:[BX]      ; Get a character from the buffer.
            INC     BX
```

```
        CMP     AL,CRCH         ; Check for Carriage Return
        JNE     BUFG1
        MOV     AH,0FFh
        MOV     CRTXT,AH        ; Set Carriage Return Flag
        JMP     GOOD
BUFG1:  CMP     AL,LFCH         ; Check for Line-Feed
        JNE     BUFG2
        MOV     AH,CRTXT        ; LF seen. Was it preceded by a CR ?
        OR      AH,AH
        JZ      GOOD
        XOR     AH,AH           ; Yes. Ignore LF after CR,
        MOV     CRTXT,AH        ; clear the flag,
        JMP     BUFGET          ; and get the next character.
BUFG2:  XOR     AH,AH
        CMP     AL,1Ah          ; Check if Ctl-Z (End-of-File)
        JE      BUFG4
        CMP     AL,TABCH
        JNE     BUFG3
        MOV     AL,20h          ; Change Tab to space
BUFG3:  OR      AL,AL
        JNE     GOOD
BUFG4:  MOV     AL,CRCH         ; Print a Carriage Return (&LF)
        CALL    COUT
        MOV     AL,LFCH
        CALL    COUT
        XOR     BX,BX           ; Clear INPTR
        MOV     AX,BX           ; Restore the Null
GOOD:   MOV     INPTR,BX
        XOR     AH,AH
        RET  Handle special character problems & test for delimiters.
QCR:    CMP     AL,DELIM
        JE      SETRET
        CMP     AL,CRCH         ; ? Carriage Return
        JE      SETRET
        CMP     AL,1Ch          ; Ctl-Z End-of-file character
        JE      SETRET
        CMP     AL,0            ; Null at end of Text-buffer
        JE      SETRET
        CLC                     ; Non-delimiter case
        RET
SETRET: STC
        RET
```

5.2. STRING WORDS

LaForth reserves an extra 64K byte segment above the code-data-stack segment to process text obtained from files. Many string words assume that the text is in this extra segment pointed to by ES segment pointer. However, compatibility with earlier versions requires that a segment address be given to these words, but it is discarded or replaced by the contents of ES segment pointer.

DSADDR (-- ds)
Push the Data Segment DS on the stack. CS and SS have the same value. This is the only word by which you can infer where the code-data segment is located in the physical memory map. The extra text segment is 1000H above the segment pointer returned by DSADDR.

```
        HEADER  RDDASD,D
```

```
DSADDR:  PUSH     DS
         NEXT
```

TEXT! (addr1 n -- addr2)
Top=delimiter byte, 2nd=address. Store n characters from the input stream in ascending addresses, top=delimiter, 2nd=address. Terminates when delimiter is reached. A null is stored in place of the delimiter, and its address is put on the stack.

```
         HEADER   !!TXET,T
TEXT:    POP      CX              ; Get delimiter
         POP      BP              ; Get address
         MOV      XHOLD,BP
         DEC      BP
TEXT1:   INC      BP
         MOV      N,BP
TEXT2:   CALL     NUCHAR
         MOV      BP,N
         OR       AH,AH
         JNZ      TEXT3
         CMP      AL,DELCH
         JE       TEXT5
TEXT3:   MOV      DS:[BP],AL
         CMP      CL,AL
         JNE      TEXT1
         CMP      AL,21h
         JB       TEXT4
         CALL     NUCHAR
         MOV      BP,N
         CMP      AL,DELCH
         JE       TEXT5
         CMP      AL,21h
         JB       TEXT4
         CMP      CL,AL
         JE       TEXT1
         INC      BP
         MOV      DS:[BP],AL
         JMP      TEXT1
TEXT4:   XOR      AL,AL
         MOV      DS:[BP],AL      ; Replace delimiter with null
         PUSH     BP
         NEXT
TEXT5:   CMP      BP,XHOLD
         JE       TEXT2
         DEC      BP
         MOV      N,BP
         MOV      DH,DS:[BP]
         CMP      DH,20h
         JB       TDEL6
         CALL     DELCHR
         JMP      TEXT2
TDEL6:                           ; Delete a control character by
         CALL     COUT            ; emitting the character then
         MOV      AL,"\"          ; sending a \ character.
         CALL     COUT
         JMP      TEXT2
```

(TEXT (delim --)
Obtain characters from the input stream and store them in the buffer area.

```
         HEADER   TXET(,H
```

```
STEXT:    POP     CX
STEXT1:   CALL    NUCHAR
          CMP     AL,DELCH
          JNE     STEXT2
          MOV     BX,TPTR
          CMP     BX,BOTB
          JE      STBEEP
          DEC     BX
          MOV     TPTR,BX
          MOV     AL,ES:[BX]
          CMP     AL,20h
          JAE     STEXTE
          CALL    DELCHR
          JMP     STEXT1
STEXTE:   CALL    COUT
          MOV     AX,005Ch          ; Send \ character
          CALL    COUT
          JMP     STEXT1
STEXT2: MOV       BX,TPTR
          CMP     AL,CL             ; Check if delimiter
          JE      STXTX
          MOV     ES:[BX],AL
          INC     BX
          MOV     TPTR,BX
          JMP     STEXT1
STXTX:    XOR     AL,AL             ; Delimiter found.  Replace it with Null.
          MOV     ES:[BX],AL
          NEXT
STBEEP:   MOV     AL,07
          CALL    COUT
          JMP     STEXT1
```

.NAME (cfa --)
Print out the name of the word whose address is on top. Character in the name are stored backward, from high address to low.

```
          HEADER  EMAN.,N
PNAME:    POP     BX
          SUB     BX,3
PNAME1:   MOV     AL,[BX]
          AND     AX,007Fh
          JZ      PNAME2
          CALL    COUT
          DEC     BX
          JMP     PNAME1
PNAME2:   NEXT

DELCHR:   MOV     BX,OFFSET ERASE
          JMP     TYPEM
ERASE     DB      8,20h,8,0
```

.TEXT (seg addr --)
Print text from buffer memory. Stop on a NULL. Top=start address, 2nd=segment.

```
          HEADER  TXET.,N
PTEXT:    POP     BX
          POP     AX                ; Throw away segment!
PTEXT1:   MOV     AL,ES:[BX]
PTEXT2:   OR      AL,AL
          JZ      PTEXT4
```

```
            CALL    COUT
            CMP     AL,0Dh              ; Is character CR ?
            JNE     PTEXT3
            MOV     AL,0Ah              ; Yes. Send a LF also.
            CALL    COUT
            INC     BX
            MOV     AL,ES:[BX]          ; Get next character
            CMP     AL,0Ah              ; Is it LF ?
            JNE     PTEXT2
PTEXT3:     INC     BX
            CALL    XKEYQ               ; Check for any key.
            JZ      PTEXT1
PTEXT4:     NEXT
```

-$< (addr1 addr2 -- f)

Backwards String LESS. Compare two strings. Two strings are compared for the purpose of ordering them. Note that no count is specified! At the first mis-match, the comparison stops. The flag is true (-1) if the byte at the first string is less than that at the second string. Note that addr1 must not be the same as addr2. Further note that the strings run backwards in memory.

```
            HEADER  !<$-,M
MSLESS:     STD
            JMP     SLESS1
```

$< (addr1 addr2 -- f)

String LESS. Compares two strings. Two strings are compared for the purpose of ordering them. Note that no count is specified! At the first mis-match, the comparison stops. The flag is true (-1) if the byte at the first string is less than that at the second string. Note that addr1 must not be the same as addr2.

```
            HEADER  !<$,D               ; WATCH MACRO CALL ***
SLESS:      CLD
SLESS1:     POP     AX
            POP     BX
            PUSH    SI
            PUSH    DI
            MOV     CX,-1
            MOV     SI,BX
            MOV     DI,AX
            CMPSB
            REPE    CMPSB
            SBB     AX,AX
            CLD
            POP     DI
            POP     SI
            PUSH    AX
            NEXT
```

5.3. THE WORD PARSER

-WORD (char – addr)

Gets Null and a reverse character string to dictionary. Leaves address of highest byte plus 1 on top of stack. There is a null at each end of the word. Top item is the delimiting character. Put a Null in the dictionary. Fetch characters from the input stream, skipping initial occurrences of the delimiter character. The non-delimiter characters are stored in the dictionary in reverse order until a delimiter or a carriage return character is encountered. Add a Null after the string in the dictionary, and return the address of that null on the stack. The dictionary pointer is not updated.

```
            HEADER  DROW-,M
```

```
MWORD:  POP     BX              ; Get delimiter
        MOV     DELIM,BL        ; Save it
        CALL    GETW
        PUSH    BX
        NEXT
GETW:   XOR     AX,AX           ; Put initial null on the stack
GETW1:  PUSH    AX
GETW2:  CALL    NUCHAR
        CALL    QCR
        JC      GETW2           ; Ignore if a delimiter
TRUB:   CMP     AL,DELCH        ; Test rub-out
        JNE     SC              ; No: It's a stack character
        POP     AX
        CMP     AX,0
        JE      GETW1           ; Put null back if at end
        CALL    DELCHR
        JMP     SC1             ; Proceed with next character
SC:     PUSH    AX
SC1:    CALL    NUCHAR
        CALL    QCR
        JNC     TRUB            ; Character string on stack
        MOV     BX,DICT         ; Get dictionary pointer  Logic to force Even boundaries for
words goes here.
        XOR     AL,AL
        MOV     [BX],AL         ; Force string terminator
        XOR     CX,CX
        DEC     CX              ; Character count set to -1
SC2:    INC     BX              ; Point BX to next character position.
        INC     CX              ; Increment character count
        POP     AX
        MOV     [BX],AL         ; Store backwards in dictionary
        AND     AL,AL
        JNE     SC2 STRING IS IN DICT (A)=0
        MOV     ARGCNT,CX       ; Save character count
        MOV     ARGLOC,BX       ; Save pointer to null at end of argument
        RET
```

SKIP (addr1 char -- addr2)
Skip over leading occurrences of char at the string beginning at addr1. Leave the address addr2 which points to the first character not equal to char .

```
        HEADER  PIKS,S
SKIP:   POP     CX              ; We use this simple approach merely
        POP     BX              ; to save code space. Use of SCAS would
SK1:    MOV     AL,[BX]         ; be faster for large no. of leading
        CMP     AL,CL           ; delimiters.
        JNE     SK2
        INC     BX
        JMP     SK1
SK2:    PUSH    BX
        NEXT
```

SCAN (addr char -- addr count)
Scan the string beginning at addr until char or a delimiter occurs. The delimiter is at addr+count.

```
        HEADER  NACS,S
SCAN:   POP     DX
        POP     BX
        PUSH    BX
        XOR     CX,CX           ; Clear the count
```

```
        MOV     DH,0Dh          ; CR character to DH
SCN1:   MOV     AL,[BX]         ; Get character
        CMP     AL,DL           ; Compare with delimiter
        JE      SCN2
        CMP     AL,DH           ; Compare with CR
        JE      SCN2
        INC     CX
        INC     BX
        JMP     SCN1            ; Loop if not a delimiter
SCN2:   PUSH    CX              ; Reached end of string
        NEXT
```

5.4. DICTIONARY SEARCH WORDS

?DEF (-- n)
Search for previously scanned word in GROWING dictionary. Searches the GROWING vocabulary for the word just obtained from the input stream. If the word is found, return the address of the word. If the word is not found, return a 0.

```
        HEADER  FED?,1F
QDEF:   MOV     BX,GROWNG
        CALL    FIND
        JMP     PTIC2
```

(' (-- n)
Returns with top=the execution address, if found If not found, top = 0 This is a primitive version of ' . When executed, get the next word from the input stream and search the dictionary for a match. If found, return the execution address on the top of the stack. If not found, return a value of 0.

```
        HEADER  !'(,H           ; WATCH MACRO CALL ***
DPX     EQU     N
PTIC:   MOV     DL,BLCH         ; Space is delimiter
        MOV     DELIM,DL
        CALL    GETW
        MOV     DX,OFFSET SRCHNG ; Pointer to vocabulary
PTIC1:  MOV     BP,DX
        MOV     BX,DS:[BP]
        OR      BX,BX
        JZ      SETF            ; Test for end of search order
        CMP     BX,DS:[BP+2]    ; See if we've searched this before
        JE      PTIC3
        CALL    FIND            ; Returns with condition Z=0 if found
PTIC2:  JNE     PUSHB
PTIC3:  SUB     DX,2
        JMP     PTIC1
PUSHB:  ADD     BX,3            ; Bump BX to execution address
        MOV     AX,BX
        JMP     XPUSH
SETF:   XOR     AX,AX           ; Set false flag
XPUSH:  PUSH    AX
        NEXT
```

Search the dictionary. (BX) = address of pointer to start of dictionary thread. ARGLOC contains address of word we are hunting for. Z=0 if found.

```
FIND:   PUSH    SI              ; Save various registers
        PUSH    DI
        PUSH    ES              ; We may not need to save ES
```

```
            STD
            MOV     AX,DS
            MOV     ES,AX
SVOC:       OR      BX,BX               ; Check if address is 0 (terminate).
            JZ      FCOM
            MOV     BL,[BX]             ; Get vocabulary number.
            MOV     DI,ARGLOC
            ADD     BL,[DI-1]           ; Add first character
            AND     BX,001Fh            ; Knock off high order bits
            SHL     BX,1                ; Multiply by 2 for word offset
            ADD     BX,OFFSET VOCABT    ; Add base of Vocabulary table
            MOV     BX,[BX]
            MOV     DPX,BX              ; Fake thread to start
            MOV     BX,OFFSET DPX-1     ; Pointer to dictionary
SDIC:       MOV     CX,ARGCNT           ; Get search count
            MOV     DI,ARGLOC           ; Get search argument
            DEC     DI
            MOV     BX,[BX]+1           ; Point SI to dictionary thread
            OR      BX,BX
            JZ      FCOM                ; If thread is 0, we can't find it.
            MOV     SI,BX
            LODSB
            AND     AL,7Fh              ; Knock off immediate bit
            JE      FOUND               ; Length=0 is universal find
            SCASB                       ; Compare with first argument character
            JNE     SDIC
            REPE    CMPSB               ; Compare remaining characters
            JNE     SDIC
FOUND:      OR      AX,1                ; Set Z=0
FCOM:       CLD                         ; Restore direction flag
            POP     ES                  ; Restore various registers
            POP     DI
            POP     SI
            RET
```

'PRE (vndx1 cfa1 -- vndx2 cfa1 cfa2)
Pushes address of proceeding dictionary word. Initially top must be a word address. At end of a dictionary thread, addr2 has a value of 0 .

```
            DB      0
            DB      "ERP'"
            CHAIN   G
TICPRE: MOV         BX,SRCHNG
            MOV     CX,[BX]             ; Get Searching Vocabulary number.
            POP     BX                  ; CFA
            POP     DX                  ; Vocabulary Index
            MOV     BP,BX               ; Save initial CFA
            SUB     BX,2                ; Point to link
TP1:        MOV     BX,[BX]             ; New head pointer
            OR      BX,BX               ; Set flags
            JE      EOCC                ; End of current chain
            MOV     AL,DL               ; Vocab Index
            SUB     AL,[BX]             ; Subtract first character
            AND     AL,01Fh             ; This word's Vnum
            CMP     AL,CL               ; Compare with Vocab Number
            JE      FND                 ; If equal, we found it.
            INC     BX                  ; Get Link
            JMP     TP1
EOCC:       INC     DX                  ; End of current chain.  Try next one.
```

```
            CMP     DL,32
            JGE     NOPRE               ; Jump if no more chains.
            MOV     BX,DX               ; New Vocab Index
            SHL     BX,1
            ADD     BX,OFFSET VOCABT    ; New Head Pointer
            JMP     TP1
FND:        ADD     BX,3                ; Get to code address
NOPRE:      PUSH    DX
            PUSH    BP
            PUSH    BX
            NEXT
```

'LAST (-- cfa)
Pushes the address of the last word in the GROWING vocabulary.

```
            DB      0
            DB      "TSAL'"
            CHAIN   G
FLAST:      MOV     BX,[LASTW]
            ADD     BX,3
            PUSH    BX
            NEXT
```

' (-- cfa) [(' 0[BACK]]
Read the next word from the input stream and pushes the address of that word onto the stack. Searches first the SEARCHING , then the ROOT vocabulary. If the string is not found, the bell rings and the cursor is backed up to the beginning of the input string. This continues until a string is found. If necessary to get out of this, use something like: DUP DROP

```
            HEADER  !',G                ; WATCH MACRO CALL ***
TIC:        NEST
            DW      PTIC
            DW      XDUP
            DW      ZBRAN
            DW      TIC1
            DW      UNNEST
TIC1:       DW      DROP
            DW      BACK
            DW      BRAN
            DW      TIC+3
```

5.5. NUMBER CONVERSION OPERATION

(NUM (-- n 0) or (-- dbl cnt) or (-- -1)
Address of digit string in ARGLOC. If the string contains no imbedded decimal points and can fit within a 16 bit word without overflow, the string is converted, the value pushed on the stack, and a flag of 0 is additionally pushed. If the string has a decimal point and can be converted to a double precision value, that value is pushed on the stack, and a flag having a value one greater than the number of digits to the right of the decimal point is pushed on the stack. If the conversion process fails, a value of -1 is pushed on the stack. If the string contains a '$' character, the following characters are treated as decimal digits. If the string contains a '#' character, the following characters are treated as hexadecimal digits. Convert, normally using value in BASE, the ASCII string just input with -WORD . If the string begins with a $ character, use 10 as a temporary base. If the string begins with a # character, use 16 (Hexadecimal) as a temporary base. A minus sign may be used to input a negative number. If the string contains a decimal point, the string is converted to a double ;number. If the string cannot be converted, a flag of -1 is returned. If a single precision number is indicated, a flag of 0 is returned. If a double number is returned, a positive number is returned containing the number of digits to the right of the decimal point, plus 1 .

```
            HEADER  MUN(,H
```

```
NUMB:   PUSH    SI              ; Save registers for other use
        PUSH    DI
        STD                     ; Setup for backward strings
        MOV     SI,ARGLOC       ; Get search argument
        DEC     SI              ; Point to first character
        XOR     BX,BX           ; Clear Accumulator
        MOV     CX,BX
        MOV     DPT,BH          ; Clear double precision flag
        MOV     EFLAG,BH        ; Clear Error flag
        MOV     DI,CBASE        ; Set current base
CB:     XOR     AH,AH
        MOV     BYTE PTR N+1,AH ; Sign switch
GDIG:   CMP     SI,DICT         ; ? Done
        JE      FINI
        LODSB                   ; Get character
        SUB     AL,'0'          ; Reduce to possible digit
        JC      LOW
        CMP     AL,10
        JC      DIGIT
        SUB     AL,7            ; Possible letter form
        CMP     AL,10           ; Invalid between 9 and A
        JC      BAD
        MOV     AH,DPT
        OR      AH,AH
        JZ      DIGIT           ; Test for Decimal point seen
        INC     DPT             ; Yes, increment count.
DIGIT:  XOR     AH,AH           ; ?Larger than base
        CMP     AX,DI
        JNC     BAD
        MOV     BP,AX           ; Current digit to BP
        MOV     AX,DI           ; Previous Accumulation to AX
        MUL     CX              ; Accum * Base
        ADD     AX,BP           ; Add in the digit
        ADC     DX,0
        MOV     CX,AX           ; Low part of new Accum
        MOV     BP,DX           ; Partial product
        MOV     AX,DI
        MUL     BX              ; Hi Accum * Base
        ADD     AX,BP           ; Hi product
        MOV     BX,AX           ; Hi part of new Accum
        ADC     DX,0
        JZ      GDIG            ; Check for overflow
BAD:
        POP     DI              ; Restore DI, SI, and DF
        POP     SI
        CLD
BAD1:   MOV     AX,-1           ; Push a "bad" flag
        PUSH    AX
        NEXT
LOW:
        ADD     AL,2            ; Is character a Decimal Point?
        JNE     TMINUS
        INC     AL
        MOV     DPT,AL          ; Set Double Flag
        JMP     GDIG
TMINUS: INC     AL
        JNE     TDOLAR
        ROR     N+1,1           ; Set Negation flag
```

```
        JMP     GDIG
TDOLAR: ADD     AL,9            ; $ forces Decimal temporary base
        JE      DECMAL
        INC     AL              ; # forces Hex temporary base
        JNE     BAD
        MOV     DI,16
        JMP     CB
DECMAL: MOV     DI,10
        JMP     CB
FINI:   POP     DI
        POP     SI
        CLD
        MOV     AL,DPT
        OR      AL,AL
        JZ      SINGLE          ; ? Single Precision
        MOV     AL,BYTE PTR N+1
        OR      AL,AL
        JZ      DDONE
        OR      BX,BX
        JS      BAD1            ; It's an error if already negative
        NEG     CX
        JNC     NEGB
        XOR     BX,-1
DDONE:  PUSH    CX
        PUSH    BX
        MOV     AL,DPT
        XOR     AH,AH
        PUSH    AX
        NEXT
NEGB:   NEG     BX
        JMP     DDONE
SINGLE: OR      BX,BX
        JNZ     BAD1            ; ? Overflow
        MOV     AL,BYTE PTR N+1
        OR      AL,AL
        JZ      SDONE
        OR      CH,CH
        JNS     SDONE
        NEG     CX
SDONE:  PUSH    CX
        XOR     AX,AX
        PUSH    AX
DONE:   NEXT
```

5.6. CURSOR BACKUP

(B (--)

Back up the cursor by one word. This is the principal error handling routine which moves the cursor back to the beginning of the word just entered. It is called when this word is not found in the dictionary and it cannot be converted to a number. LaForth does not prompt you with 'ok', as most Forth does. If it accepts a word, the word is processed (executed or compiled) immediately. It will only inform you that it fails to process a word by beeping and backing up the cursor.

```
        HEADER  B(,H
BACK:   MOV     AX,7            ; Bell code
        CALL    COUT
```

```
        MOV     CX,ARGCNT
        INC     CX              ; Account for delimiter
        MOV     BX,ARGLOC
BNL:    MOV     AX,BSCH         ; Backspace character
        CALL    COUT            ; Output backspace
NOBS:   DEC     BX              ; Point to next character
        MOV     AL,[BX]         ; Examine next character
        CMP     AL,020h
        JGE     PBK
        DEC     CX
        JZ      PBK2
        JMP     NOBS
PBK:    LOOP    BNL
PBK2:   INC     EFLAG           ; Set error
        MOV     BX,INPTR        ; Print out if error in Run Mode
        OR      BX,BX
        JE      DONE
        SUB     BX,ARGCNT
        DEC     BX              ; Don't forget delimiter
        XOR     AX,AX
        MOV     INPTR,AX        ; Clear Run Mode
        MOV     CSTATE,AX       ; Clear Compile state
        JMP     PTEXT1
```

5.7. TEXT INTERPRETER

INTERPRET (--)

Process one input word, compile if STATE is true. Input a string from the input stream and interpret it. If STATE is 0, execute it; otherwise, compile it. If it is not a word in the dictionary, convert it to a number. If STATE is 0, leave the number on the stack; otherwise, compile the number as a literal into the dictionary.

```
        HEADER  TERPRETNI,I
INTERP: NEST
INT1:   DW      PTIC                    ; Find next word in input stream.
        DW      STATE,AT,ZBRAN,XEQNUM   ; If state = 0, Execute or push.
        DW      XDUP,ZBRAN,CMPLIT       ; If word not found, compile lit.
        DW      XDUP,THREE,SUB,CAT
        DW      CLIT
        DB      80h
        DW      LESS
        DW      ZBRAN,XEQIT             ; Immediate -- So execute it.
        DW      COMMA
        DW      UNNEST
CMPLIT: DW      DROP                    ; Compile a literal.Drop 0 from FIND
        DW      NUMB                    ; Literal value
        DW      XDUP
        DW      ZLESS                   ; Check if valid number
        DW      ZBRAN
        DW      CMPL1
        DW      DROP
BADNUM: DW      SRCH                    ; It's a bad number
        DW      CLIT
        DB      6
        DW      PLUS
        DW      AT
```

```
            DW      EXEC
            DW      BRAN
            DW      INT1
CMPL1:      DW      ZBRAN
            DW      CSNGL
            DW      COMP            ; It's Double Precision
            DW      DLIT
            DW      COMMA
            DW      COMMA
            DW      UNNEST
CSNGL:
            DW      XDUP            ; ? LIT or CLIT
            DW      LIT
            DW      0FF00h
            DW      XAND
            DW      ZBRAN
            DW      CCLIT
            DW      COMP
            DW      LIT
            DW      COMMA
            DW      UNNEST
CCLIT:      DW      COMP            ; Compile a Character Literal.
            DW      CLIT
            DW      CCOMM
            DW      UNNEST
XEQNUM:
            DW      QDUP            ; State is zero. Execute or make zero.
            DW      ZBRAN
            DW      MKNUM
XEQIT:      DW      FROMR           ; Execute it.
            DW      ORPH1
            DW      STORE
            DW      EXEC
            DW      QSTACK
            DW      ORPH1
            DW      AT
            DW      TOR
            DW      UNNEST
MKNUM:      DW      NUMB
            DW      ZLESS
            DW      ZEQU
            DW      ZBRAN
            DW      BADNUM
            DW      UNNEST
ORPH1:      CALL    AT              ; This is an "orphan"
            DW      RHOLD

SO          ( -- addr )
Pushes address of stack origin on the stack.
            HEADER  OS,S
SO:         NEST
            DW      LIT
            DW      TOES
            DW      AT
            DW      TWOM
            DW      UNNEST
```

RUN (seg addr --)
Transfers top to IN. Used to execute from a text buffer. Contents of the text buffer are read from low address to high and control returns to the keyboard when a NULL character is encountered. If addr is 0, take input from the keyboard. Otherwise, addr is the address offset pointing to the text stream to be interpreted in the text buffer.

```
        HEADER  NUR,R
RUN:    NEST
        DW      LIT
        DW      INPTR
        DW      STORE
        DW      DROP
        DW      UNNEST
```

?STACK (--)
Test for stack underflow, and issue "EMPTY STACK" and call QUIT. ;Also tests for stack full and reports if less than 256 bytes ;remain. You can make more stack space by forgetting from the ;dictionary or dropping words from the stack. You have 256 bytes ;to use before the stack overruns the dictionary. Equivalent Forth code is:

SO SP@ 1+ U< ?[." Empty Stack" QUIT]? MEM #FF U< ?[." MEM=" MEM .B DROP]?

```
        HEADER  KCATS?,1F
QSTACK: NEST
        DW      SO
        DW      SPAT
        DW      TWOP
        DW      ULESS
        DW      ZBRAN
        DW      Q1
RMT     DW      PTYPE           ; R-stack is empty
        DB      'Empty Stack '  ; Print message,
        DB      7,0             ; Ring the Bell.
        DW      QUIT
Q1      DW      MEM             ; Test if less than 255 bytes left
        DW      CLIT
        DB      0FFh
        DW      ULESS
        DW      ZBRAN
        DW      Q2
        DW      PTYPE
        DB      'MEM='
        DB      7,0
        DW      MEM
        DW      HPB
        DW      DROP
Q2      DW      UNNEST
```

5.8. SYSTEM INITIATION

QUIT (--)
This is where the text interpreter starts. The system is prepared to accept and process text from the keyboard. Clears the computational and R-stacks, then pushes the address of the input area, zero for keyboard, and executes CR. Equivalent Forth code is:

SP! IN @ 0 0 RUN 0 STATE ! RP! RMT 2- >R CR [[INTERPRET]]

```
        HEADER  TIUQ,Q
```

```
QUIT:   NEST
        DW      SPSTO
        DW      LIT
        DW      INPTR
        DW      AT
        DW      ZERO
        DW      ZERO
        DW      RUN
        DW      ZERO
        DW      STATE
        DW      STORE
        DW      RCLR
        DW      LIT
        DW      RMT-2
        DW      TOR
        DW      CR
QUIT1   DW      INTERP
        DW      BRAN
        DW      QUIT1
```

WARM (--)
Warm start. Issues the entry message and calls QUIT . Equivalent Forth code is:

```
DECIMAL ROOT DEFS CR ." PC LaForth V4.0" QUIT
```

```
        HEADER  MRAW,W
WARM:   NEST
        DW      DEC
        DW      ROOT
        DW      DEFS
        DW      CR
        DW      PTYPE
        DB      'PC LaForth V4.0'
        DB      0
        DW      QUIT
```

COLD (--)
Cold start. First check to see if an input file was specified on the DOS command line. If true, open the input file and read it into the text buffer. Then pass control to WARM and bring the LaForth system up.

```
        HEADER  DLOC,C
COLD:   NEST
        DW      SPSTO
        DW      CLIT
        DB      80h             ; Check if any input file specified
        DW      CAT             ; on the Command Line.
        DW      ZBRAN           ; 80h C@ IF 81h 20h SKIP 20h SCAN OVER +
        DW      COLD1           ; 0 SWAP C!  OPEN >R LT DUP NEG 110h -
        DW      DSADDR          ; R> READ TP +!  0 LT DROP TP @ XC!
        DW      CLIT            ; 0 80 C!  THEN
        DB      81h             ; WARM
        DW      CLIT
        DB      20h
        DW      SKIP
        DW      CLIT
        DB      20h
        DW      SCAN
        DW      OVER
        DW      PLUS
```

```
        DW      ZERO
        DW      SWAP
        DW      CSTOR
        DW      OPEN
        DW      TOR
        DW      LT
        DW      XDUP
        DW      XNEG
        DW      LIT
        DW      110h
        DW      SUB
        DW      FROMR
        DW      READ
        DW      TP
        DW      PLSTOR
        DW      ZERO
        DW      LT
        DW      DROP
        DW      TP
        DW      AT
        DW      XCSTOR
        DW      ZERO
        DW      CLIT
        DB      80h
        DW      CSTOR
COLD1:  DW      WARM
```