# CHAPTER 6. DATA AND CONTROL STRUCTURES

## 6.1. LITERAL STRUCTURES

LIT ( -- n )
Useful only within a : definition to get the value of the word which follows pushed onto the stack. Note the following word can not be an Immediate word, for this to work. Prefix word for In-line 16 bit literals.

```
        HEADER  TIL,L
LIT:    LODSW
        PUSH    AX
        NEXT
```

CLIT ( -- n )
A primitive word which is followed by a byte literal in a colon definition. The value of the literal is concatenated with 8 high order 0 bits and pushed on the stack when CLIT is executed. Prefix word for In-line 8 bit literals.

```
        HEADER  TILC,C
CLIT:   LODSB
        XOR     AH,AH
        PUSH    AX
        NEXT
```

DLIT ( -- d )
A primitive to be used during compilation, and typically followed by a double-word in-line literal. When DLIT is executed, the two following words are pushed onto the stack. Prefix word for In-line 32 bit literals.

```
        HEADER  TILD,D
DLIT:   LODSW
        MOV     DX,AX
        LODSW
        PUSH    AX
        PUSH    DX
        NEXT
```

## 6.2. STRING LITERAL STRUCTURES

(." ( -- )
In-line primitive for ." strings. Prints the succeeding in-line characters until a Null is encountered.

```
        DB      0
        DB      '"'
        DB      ".("
        CHAIN   H
PTYPE:  MOV     BX,SI
        CALL    TYPEM
        MOV     SI,BX
        INC     SI
        NEXT
```

(" ( – addr )
Primitive in-line string operator. The count of the string is at the word preceding the address. The string is terminated by a null (not included in the count).

```
        DB      0
        DB      '"'
        DB      '('
```

```
        CHAIN   H
PQUOTE: MOV     BX,[SI]
        ADD     SI,2
        PUSH    SI
        INC     SI
        ADD     SI,BX
        NEXT
```

## 6.3. CONTROL STRUCTURE WORDS

(=?[ ( n1 n2 -- ) or ( n1 n2 -- n1 )

Primitive CASE branch. When executed within a colon definition, the top two stack elements ar tested for equality. If the top two elements have a different value, the top element is dropped and the next in-line value is taken as the branch address. If the two elements are equal, both elements are dropped, the in-line value is skipped, and execution continues.

```
        HEADER  [?=(,H
CASE:   POP     AX
        MOV     BP,SP
        CMP     AX,[BP]
        JNE     BRAN
        POP     AX
        JMP     NOBRAN
```

(?] ( f -- )

Branch if stack is false (0) primitive conditional branch. When executed within a colon definition, the top of the stack is popped and tested. If the result is zero, the next in-line value is taken as the branch address. If the result is non-zero, the in-line value is skipped, and execution continues.

```
        HEADER  ]?(,H
ZBRAN:  POP     AX
        OR      AX,AX
        JZ      BRAN
NOBRAN:
        ADD     SI,2
        NEXT
```

(] ( -- )

Unconditional branch Test the value at the top of the return stack. If it is non-zero, decrement it by 1 and then jump to the location specified ;y the next in-line value. If the top of the Return stack is zero, pop it from the Return stack, jump over the in-line value, and continue execution.

```
        HEADER  ](,H
BRAN1   EQU     THIS WORD               ; Strange construct to allow
BRAN:   MOV     SI,[SI]                 ; modification of code !!!
        NEXT    Break-Key Branch Code
ABNORM: MOV     AX,348Bh                ; Code for  MOV SI,[SI]
        MOV     BRAN1,AX                ; Restore to BRAN
        XOR     AX,AX                   ; Clear the Break Flag
        MOV     DS,AX
        MOV     BX,471h
        MOV     [BX],AL
        MOV     AX,CS
        MOV     DS,AX
        JMP     WINIT
```

]# ( -- )
End of run-time loop. Test the value at the top of the return stack. If it is non-zero, decrement it by 1 and then jump to the location specified by the next in-line value. If the top of the Return stack is zero, pop it from the Return stack, jump over the in-line value, and continue execution.

```
            HEADER    #](,H
PLOOP:      DEC       WORD PTR ES:-2[DI]
            JNS       BRAN
            MOV       AX,ES:-2[DI]
            INC       AX
            JNZ       BRAN
            SUB       DI,2
            ADD       SI,2
            NEXT
```

## 6.4. DICTIONARY HEADER

HEAD, ( -- )
Obtain the next word from the input stream and create a dictionary entry containing the name field and linkage, but no action part of the new word. The head starts with a name field, which begins with a null byte and the name of the new word laid down backward. The name field is followed by a 2 byte link field, pointing to the link field of the link field of the previously defined word in its linked chain. Following the name field is the code field where executable code will be placed. Equivalent Forth code is:

#20 -WORD DUP DP ! 1- DUP C@ GROWING @ @ + 1F AND
2* VOCABT + DUP @ , OVER LATEST ! !

```
            HEADER    !,DAEH,H
HEADC:      NEST
            DW        CLIT
            DB        20h
            DW        MWORD
            DW        QDEF
            DW        QDUP
            DW        ZBRAN
            DW        HEADC1
            DW        SPACE
            DW        PNAME
            DW        PTYPE
            DB        " previously defined ",0
HEADC1:     DW        XDUP
            DW        DP
            DW        STORE
            DW        ONEM
            DW        XDUP
            DW        CAT
            DW        GROW
            DW        AT
            DW        AT
            DW        PLUS
            DW        CLIT
            DB        1Fh
            DW        XAND
            DW        MTWO
            DW        LIT
            DW        VOCABT
```

```
            DW      PLUS
            DW      XDUP
            DW      AT
            DW      COMMA
            DW      OVER
            DW      LATEST
            DW      STORE
            DW      STORE
            DW      UNNEST
```

IMM (--)
Make the last defined word have the property that it executes when used inside a : definition rather than be compiled. Used to create the conditional and looping words.

```
            HEADER  MMI,I
IMM:        NEST
            DW      LATEST
            DW      AT
            DW      SCOMP
            DW      UNNEST
```

## 6.5. VOCABULARY

:VOC (--)
A defining word which creates a new vocabulary. Word 0 of the new vocabulary body contains a vocabulary number (in the range 0 to 31), which is incremented for each new vocabulary. Word 1 contains a link to a prior vocabulary to be searched, if a sought after word is not found in the current vocabulary. This is nominally set to ROOT . Word 2 is a vocabulary linkage used by a "smart" FORGET . Word 3 contains the address of a word to be executed when the search fails. This is normally set to (NUM to specify a search for a literal number. Word 4 contains the address of a word to be executed when the attempt to make a literal fails. This is normally set to (B , which will back up the cursor, and beep. Equivalent Forth code is:

:BUILD 1 VOCNUM +! HERE 3 - VOCNUM @ , VOCNUM 2+ DUP HERE $0E CMOVE ! $0E ALLOT 3 - INSTALL

```
            HEADER  COV!:,Z             .; WATCH MACRO CALL ***
VOC:        NEST
            DW      BUILD
            DW      ONE
            DW      VOCNUM
            DW      PLSTOR
            DW      HERE
            DW      THREE
            DW      SUB
            DW      VOCNUM
            DW      AT
            DW      COMMA
            DW      VOCNUM
            DW      TWOP
            DW      XDUP
            DW      HERE
            DW      CLIT
            DB      0Ch
            DW      CMOVE
            DW      STORE
            DW      CLIT
```

```
            DB      0Ch
            DW      ALLOT
            DW      PSEMIC
DOVOC:      LCALL   DODOES
            DW      THREE
            DW      SUB
            DW      INSTAL
            DW      UNNEST
```

DEFS ( -- )
Make GROWING have the value of SEARCH.

```
            HEADER  SFED,D
DEFS:       NEST
            DW      SRCH
            DW      AT
            DW      GROW
            DW      STORE
            DW      UNNEST
```

INSTALL ( voc-addr -- )
Install the vocabulary specified by the address in the SEARCHING vocabulary.

3 + DUP SEARCHING ! 2+ SEARCHING 2+ $0E CMOVE

```
            HEADER  LLATSNI,I
INSTAL:     NEST
            DW      THREE
            DW      PLUS
            DW      XDUP
            DW      SRCH
            DW      STORE
            DW      TWOP
            DW      SRCH
            DW      TWOP
            DW      CLIT
            DB      0Eh
            DW      CMOVE
            DW      UNNEST
```

ROOT ( -- )
The basic default vocabulary. WARM establishes this as both SEARCH and GROWING vocabularies.

```
            DB      0
            DB      'TOO'
            DB      'R' OR IMMFLG       ; IMMEDIATE
            CHAIN   R
ROOT:       CALL    DOVOC               ; ROOT
            DW      0                   ; Vocabulary Number
            DW      0                   ; Vocabulary link
            DW      NUMB                ; Not in dictionary
            DW      BACK                ; Not valid number
            DW      0                   ; Spare (What to do on Break?)
            DW      0                   ; Spare
            DW      0                   ; Spare
            DW      0                   ; Spare
```

## 6.6. THE VOCABULARY LINKS

VOCTABLE ( – addr )
Address of the vocabulary table. This table contains 32 addresses, which point to the last words defined in the 32 vocabulary threads. LaForth hashes words and vocabularies into 32 threads. Each vocabulary is assigned a vocabulary index. This index is added to the first character of a word to be linked into this vocabulary. The sum is multiplied by 2, forming an offset into this vocabulary table to select one of the threads to which the new word is added. Thus a word can be uniquely identified by its name and the vocabulary it belongs.

```
          HEADER    ELBATCOV,V
VOCTAB:   LCALL     AT
          DW        VOCABT
```

The threads are constructed using the following macro facilities in MASM assembler, at the very beginning of the source listing.

```
HEADER    MACRO     text,sfx            ;; For creating headers
          DB        0                   ;; Backwards terminator is a null.
          DB        "&text"
          DW        LINK&sfx-3
          LINK&sfx = $
ENDM

CHAIN     MACRO     sfx                 ;; Auxiliary for making headers when
          DW        LINK&sfx-3          ;; the text is difficult for Assembler.
          LINK&sfx = $
ENDM

LINK0 = 3                               ;; First of 32 links

IRPC      sfx,ABCDEFGHIJKLMNOPQRSTUVWXYZ        ;; Next 26 links
          LINK&sfx = 3
ENDM

IRPC      sfx,BCDEF                     ;; Last 5 links.

          LINK1&sfx = 3
ENDM
```

These assembly macros establish 32 linked lists, with names LINK0, LINKA to LINKZ, and LINK1B to LINK1F. The first link in each list is 0, which tells the text interpreter it is the end of a linked thread. The last link of a thread is store in a table VOCABT, which is defined at the end of the source listing.

At the end of the source listing, the vocabulary link table is allocated after the word ROOT.

```
VOCABT    DW        LINK0-3             ; Vocabulary table

          IRPC      sfx,ABCDEFGHIJKLMNOPQRSTUVWXYZ
          DW        LINK&sfx-3
ENDM
```

The assembler resolves the link addresses for the 32 threads and places these link heads in VOCABT table:

```
3391    149E  109B R     +          DW      LINKA-3
3392    14A0  1294 R     +          DW      LINKB-3
3393    14A2  142B R     +          DW      LINKC-3
3394    14A4  13A3 R     +          DW      LINKD-3
3395    14A6  1065 R     +          DW      LINKE-3
```

```
3396        14A8  0BFB R          +          DW        LINKF-3
3397        14AA  1014 R          +          DW        LINKG-3
3398        14AC  12EC R          +          DW        LINKH-3
3399        14AE  1419 R          +          DW        LINKI-3
3400        14B0  0C69 R          +          DW        LINKJ-3
3401        14B2  06B0 R          +          DW        LINKK-3
3402        14B4  1284 R          +          DW        LINKL-3
3403        14B6  1266 R          +          DW        LINKM-3
3404        14B8  0849 R          +          DW        LINKN-3
3405        14BA  122E R          +          DW        LINKO-3
3406        14BC  062E R          +          DW        LINKP-3
3407        14BE  134F R          +          DW        LINKQ-3
3408        14C0  1486 R          +          DW        LINKR-3
3409        14C2  1207 R          +          DW        LINKS-3
3410        14C4  125C R          +          DW        LINKT-3
3411        14C6  1071 R          +          DW        LINKU-3
3412        14C8  1032 R          +          DW        LINKV-3
3413        14CA  137D R          +          DW        LINKW-3
3414        14CC  06FC R          +          DW        LINKX-3
3415        14CE  0000            +          DW        LINKY-3
3416        14D0  13DB R          +          DW        LINKZ-3

            IRPC      sfx,BCDEF
            DW        LINK1&sfx-3
ENDM
;
3420        14D2  0F89 R          +          DW        LINK1B-3
3421        14D4  05CB R          +          DW        LINK1C-3
3422        14D6  0F5B R          +          DW        LINK1D-3
3423        14D8  0558 R          +          DW        LINK1E-3
3424        14DA  12A8 R          +          DW        LINK1F-3
```