FORTH INTRODUCTION

1. PREFACE

SUMMARY

- WHAT IS FORTH ?
- WHY CHOOSE FORTH ?
- K1197 / K1297 IMPLEMENTATION OF FORTH
- FORTH ENHANCEMENTS

WHAT IS FORTH ?

A PROGRAMMING LANGUAGE

- HIGH LEVEL
- LOW LEVEL

AN OPERATING SYSTEM

- INTERPRETER
- COMPILER
- ASSEMBLER
- TEXT-EDITOR
- MEMORY MANAGEMENT
- I/O MANAGEMENT

A PHILOSOPHY

- FUNCTIONAL STRUCTURE
- MODULAR
- INFORMATION HIDING
- PROTOTYPING

WHY CHOOSE FORTH ?

- INTERACTIVE
- EXTENSIBLE
- STRUCTURED
- FAST
- COMPACT

K1197 IMPLEMENTATION OF FORTH

CONTAINS ELEMENTS OF

- FORTH-79
- FIG-FORTH
- 68K-FORTH

EXTENSIONS AND ENHANCEMENTS

DISK FILE SYSTEM :
HIGH SPEED FILE SYSTEM TO HANDLE VARIABLE LENGTH NAMED FILES

EDITOR :
COLOR EDITOR

DISPLAY :
COLOR SCREEN
MULTIPLE CHARACTER SETS
WINDOW OPERATIONS
ENTRY OF UNPRINTABLE CHARACTERS

TERMINAL INPUT :
FUNCTION KEY CAPABILITY
REMOTE TERMINALS

DATA COMMUNICATION :
TRANSMISSION AND RECEPTION OF SERIAL DATA
TIMER MANIPULATION :
SEPARATE CONFIGURATION OF EACH TEST PORT

REAL TIME DISK RECORDING :
DATA STREAM MERGER DSM

CASE CONSTRUCT :
SELECT ONE OF A SET OF POSSIBLE CASES

FORTH ENHANCEMENTS

- EXTENSIONS FOR
  CONTROL STRUCTURES AND VECTORED EXECUTION

- EXTENSIONS FOR
  COMFORTABLE USER INTERFACE (MENU SYSTEM, FUNCTION KEYS)

- EXTENSIONS FOR PROTOCOL DECODING AND TESTING

2. FORTH CONCEPTS

SUMMARY

- WORD
- DICTIONARY
- INTERPRETER
- RPN - REVERSE POLISH
- STACK
- COMPARATIVE OPERATORS AND BOOLEAN FLAGS

WORD

UNIT OF EXECUTION (PROCEDURE, FUNCTION, SUBROUTINE)

- EVERY ELEMENTS OF THE LANGUAGE FORTH IS A **WORD** (EXCEPT LITERALS)

- EXECUTABLE PROCEDURE
  EXECUTED BY SIMPLY CALLING ITS NAME

- MAY BE DEFINED AS A GROUP OF ALREADY EXISTING WORDS

- DEFINITION OF A **WORD** LEAVES A **DICTIONARY** ENTRY

- INFORMATION IS USUALLY PASSED TO AND FROM A **WORD** ON THE **PARAMETER STACK**

DICTIONARY

CONTAINS DEINITIONS OF WORDS

- THREADED LIST OF **WORDS**

- ONCE A **WORD** HAS BEEN ENTERED IN THE **DICTIONARY** (DEFINED), EXECUTED INTERPRETIVELY

- **DICTIONARY** ENTRIES ARE DONE BY DEFINING **WORDS**

- **WORDS** ARE THREADED IN THE ORDER THEY ARE DEFINED. NOT IN LEXICAL ORDER

- THE FIRST PART OF THE DICTIONARY IS THE SYSTEM DICTIONARY (LIKE A LIBRARY OF USEFUL COMMANDS)

INTERPRETERS

IN FORTH THERE ARE 2 TYPES OF INTERPRETERS TO BE DISTINGUISHED :

OUTER INTERPRETER

- IDENTIFIES A **WORD** IN THE DICTIONARY USING THE HEADER OF THE WORD

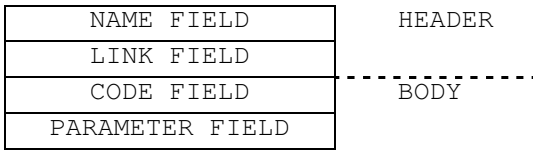
INNER INTERPRETER

- EXECUTES THE WORD **USING ITS BODY**

OUTER INTERPRETER

REMEMBER EACH FORTH **WORD** HAS A DICTIONARY ENTRY.

THE STRUCTURE IS THE FOLLOWING :



THE OUTER INTERPRETER IS A LOOP WHICH COMPASRES STRINGS IN THE INPUT STREAM WITH THE NAMES OF FORTH WORDS IN THE DICTIONARY.

THE LATEST DICTIONARY ENTRY IS COMPARED FIRTST AND THAN THE PREVIOUS ON USING THE LINK POINTER FIELD AND SO ON.

IF A MATCH IS FOUND, CONTROL IS PASSED TO THE INNER INTERPRETER.

OTHERWISE, THE STRING IS TRIED TO BE INTERPRETED AS A NUMBER, WHICH IS PLACED ON TOP OF STACK (TOS).

IF THIS IS NOT POSSIBLE, SINCE ANY CHARACTER IS NOT A VALID DIGIT ACCORDING TO THE CURRENT **BASE**, THE STRING IS DISPLAYED AT THE SCREEN FOLLOWED BY ??.

INNER INTERPRETER

THE CODE POINTER IN THE BODY OF A **WORD** POINTS TO THE CODE TO BE EXECUTED USING THE _ IN THE PARAMETER FIELD.

THE PARAMETERS MAY BE POINTERS TO OTHER **WORDS** BY WHICH THE CURRENT **WORD** IS DEFINED. TO CONTOL THIS NESTED EXECUTION IS THE JOB OF THE **INNER INTERPRETER.**

RPN – REVERSE POSISH NOTATION (POSTFIX NOTATION)

PHILOSOPHY

- PARAMETERS ARE NOTED FIRST FOLLOWED BY THE OPERATOR(S)

- THE INPUT IS UNAMBIGUOUS WITHOUT BRACKETS.

EXAMPLES:

INFIX	PRN
2 + 4 =	2 4 +
2 / (4+3) =	2 4 3 + /
2 / 4 + 3 =	2 4 / 3 +

THERE IS A **FORTH WORD** EXISTENT FOR EACH ARITHMETICAL OPERATOR.

RPN IN FORTH MEANS THAT ALL PARAMETERS OF A **FORTH WORD** HAVE TO BE ON THE PARAMETER STACK BEFORE THE WORD IS EXECUTED.

STACK

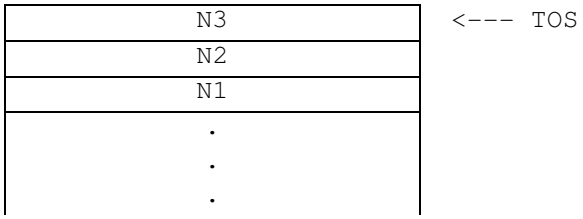
PARAMETER STACK

- CENTRAL INSTRUMENT FOR PARAMETER PROCESSING

- MAKES MANY AUXILIARY VARIABLES SUPERFLUOUS

- ORGANIZED AS **LIFO** (LAST IN - FIRST OUT) BUFFER

- EACH ENTRY CONTAINS 4 BYTES



- EACH ENTRY OF EXTRACTION OF THE TOS - ELEMENT CHANGES A STACK-POINTER

RETURN STACK

- LIFO ORGANIZED (LIKE PARAMETER STACK)

- USED FOR :
  - LOOP OPERATIONS
  - SAVING OF RETURN ADDRESSES IN NESTED OPERATIONS
  - TEMPORARY STORING OF PARAMETERS (CAUTIONS ! !)

>R PUT VALUE ONTO RETURN STACK
R MAKE A COPY OF TOP OF RETURN STACK
R> TAKE VALUE FROM RETURN STACK

STACK NOTATION OF FORTH WORDS

- GENERAL FORM:
  (BEFORE EXECUTION --- AFTER EXEC.)

- FOR MULTIPLE STACK ENTRIES "\" IS USED TO SEPARATE THE ENTRIES.

- THE LEFT-HAND ENTRY IS THE LOWEST ITEM ON THE STACK, THE RIGHT-HAND THE HIGHEST

EXAMPLES :

(V1\V2 -- V3)

READ AS:
EXPECTS V1 UNDER V2, LEAVES V3

EXAMPLES :

```
* (N1\N2  --- P)
```

THE WORD * EXPECTS TWO VALUES ON THE STACK, THE TWO NUMBERS TO BE MULTIPLIED AND LEAVES THE RESULT OF THIS MULTIPLICATION.

```
USAGE : 3 4 * . CR
        12 OK
```

STACK DESCRIPTION

(MULTIPLICAND 1 \ MULTIPLICAND 2 --- PRODUCT)

PARAMETER STACK OPERATIONS

- INPUT
- OUTPUT
- DUP
- SWAP
- ROT
- OVER
- PICK
- DROP

STACK INPUT

THE PARAMETER STACK IS FILLED BY
- KEYBOARD ENTRIES OF NUMBERS OR LITERALS
  (ONLY DIGITS ACCORDING TO THE CURRENT **BASE** ARE ALLOWED)
- CALLING **FORTH WORDS** WHICH LEAVE ENTRIES
  (THE NUMBER OF ENTRIES A **WORD** LEAVES IS NOT FIXED)

STACK OUTPUT

THE PARAMETER STACK IS EMPTIED BY CALLING FORTH WORDS WHICH EXTRACT **STACK** ENTRIES
STACK SIZE IS 200 ENTRIES(200 * 4 BYTES)

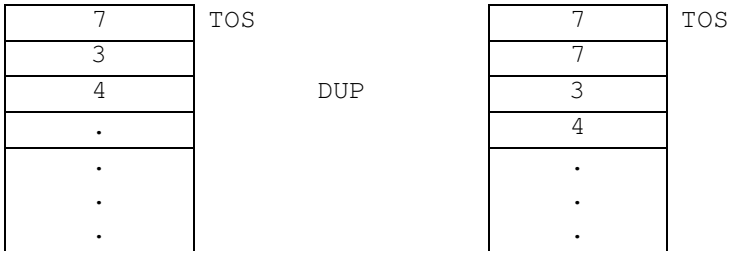
.	REMOVES TOS ENTRY AND OUTPUTS IT AS VLAUE ACCORDING TO THE CURRENT BASE
S.	OUTPUTS TOS ENTRY WITHOUT MODIFYING THE STACK
.S	OUTPUTS TOP 4 STACK ENTRIES WITHOUT MODIFYING THE STACK
STACK	OUTPUTS COMPLETE STACK WITHOUT MODIFYING IT
SP!	CLEAR THE STACK

DUP (N1 -- N1\N1)

DOUBLES TOS ENTRY

EXAMPLES:

```
4 3 7 CR OK
DUP  CR  OK
....  CR  7 7 3 4 OK
```



SWAP (N1\N2 --- N2\N1)

SWAPS THE TWO TOP ENTRIES

EXAMPLES:

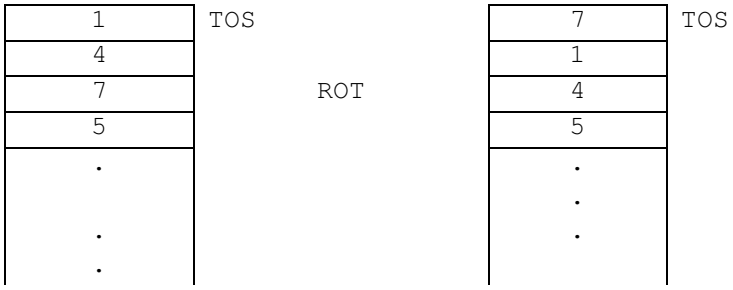
```
4  8  3  CR   OK
SWAP     CR   OK
. . .    CR   8 3 4  OK
```

ROT (N1\N2\N3 --- N2\N3\N1)

ROTATES 3 ENTRIES SUCH THAT THE THIRD ENTRY LIES ON TOP

EXAMPLES:

```
5 7 4 1  CR  OK
ROT      CR  OK
. . . .  CR  7 1 4 5  OK
```

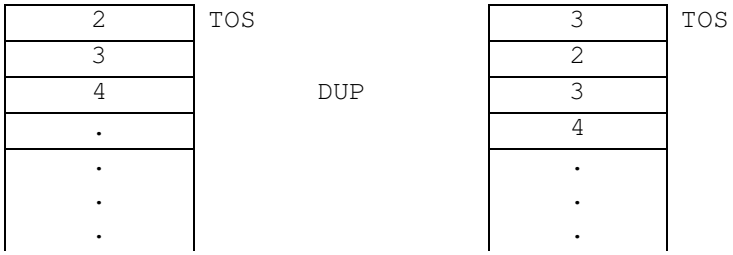


OVER (N1\N2 --- N1\N2\N1)

COPIES THE VALUE BELOW TOS ON TOS

EXAMPLES :

```
4  3  2   CR  OK
OVER       CR  OK
. . . .    CR  3 2 3 4 OK
```

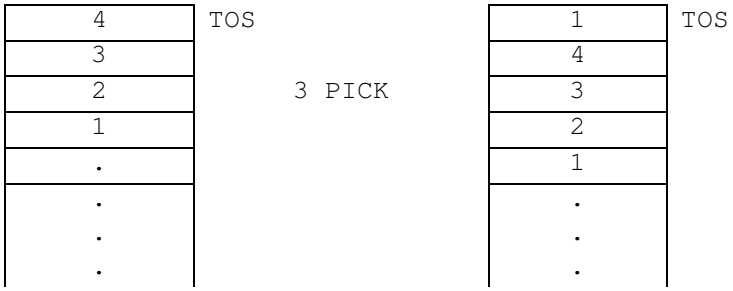


PCIK (N --- M)

LEAVES A COPY (M) OF THE Nth STACK ENTRY ON TOS
N STARTS AT O, THUS 0 PICK IS THE SAME AS DUP. 1 PICK IS THE SAME AS OVER.

EXAMPLES:

```
1 2 3 4   CR   OK
3 PICK    CR   OK
. . . . . CR   1 4 3 2 1 OK
```



DROP (N --)

DROPS TOS ENTRY

```
5  7  2   CR   OK
```

EXAMPLES:

```
DROP      CR   OK
. . .     CR   7  5  STACK EMPTY  OK
```

COMPARATIVE OPERATORS AND BOOLEAN FLAGS

THE OPERATOR " < " (LESS THAN) MAY SERVE AS AN EXMPLES
(D1 \ D2 -- BOOLEAN FLAG)

A BOOLEAN FLAG REPRESENTS ONE OF THE LOGICAL VALUES "TRUE" OF "FALSE"

THE REPRESENTATION OF "TRUE" IS 1 (OR ANY NONZERO NUMBER) THE REPRESENTATION OF "FALSE" IS 0

EXAMPLE:

```
1 2 < .  CR  1 OK
2 1 < .  CR  O OK
```

3. FORTH DEFNINITION

SUMMARY

- GENERAL
- HOW TO DEFINE NEW WORDS
- MODULAR CODING

GENERAL

IN FORTH, ALL COMMANDS ARE CALLED WORDS. EACH **WORD** IS EXECUTED BY SIMPLY CALLLING ITS NAME (IF THE WORD NEEDS PARAMETERS THEY ARE EXPECTED TO BE ON THE PARAMETER STACK BEFORE THE WORD IS CALLED).

EXAMPLES:

```
4  15  SPACES . CR

---------------4 OK
```

SPACES (NUMBER --)

THE WORD **SPACES** MOVES THE CURSOR TO THE RIGHT BY A NUMBER OF POSITIONS TAKEN FROM THE STACK.

EXAMPLES:

```
." I AM LEARNING FORTH"  CR
I AM LEARNING FORTH  OK
```

THE WORD ." OUTPUTS THE FOLLOWING STRING. IT MUST BE SEPERATED FROM THE STRING BY ONE BLANK.

STRING: I AM LEARNING FORTH

DELIMETER OF " (THIS IS NOT A FORTH WORD)
THE STRING:

NOTE:

NAMES OF **FORTH WORDS** MAY CONSISTS OF UP TO 31 CHARACTERS.
THEY MUST BE SEPERATED FROM ONE ANOTHER BY AT LEAST ONE BLANK.

HOW TO DEFINE NEW WORDS

FORTH IS AN EXTENSIBLE LANGUAGE

YOU MAY USE ALREADY DEFINED WORDS TO EXTEND THE DICTIONARY BY DEFINING NEW WORDS.

EXAMPLE:

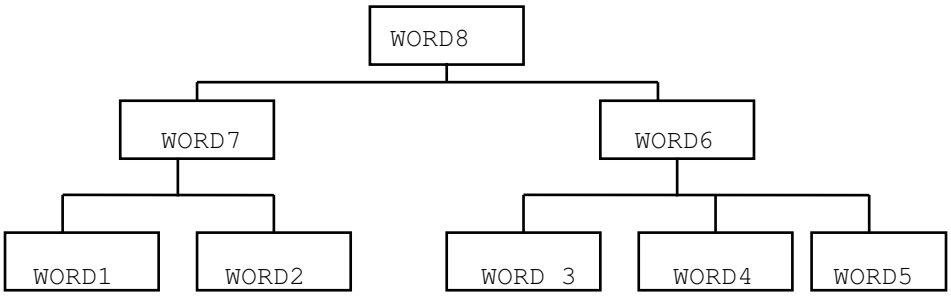
```
: NEW_WORD ." LEARNING FORTH" ;
|   |      |    |           | |
|   |      |    |           | +-- WORD TERMINATED
|   |      |    |           +---- TEXT DELIMITER
|   |      |    +---------------- TEXT
|   |      +--------------------- F. WORD "DISPLAY TEXT"
|   +-------------------------- NAME OF THE WORD
+------------------------------ DEFINING WORD
```

MODULAR CODING

FORTH ENCOURAGES **MODULA** CODING.

A NEW **WORD** IS DEFINED BY USING PREVIOUSLY DEFINED WORDS.

SO YOU CAN DEFINE "COMPLICATED" **WORDS** BY "LESS COMPLICATED" **WORDS** AND THESE BY EVEN LESS COMPLICATED **WORDS**.



BUT NOTE:

A **WORD** MUST BE DEFINED BEFORE IT IS CALLED OR USED IN A DEFINITION.

SO, THE DESIGN OF A **FORTH PROGRAM** MAY BE TOP-DOWN, FROM GENERAL TO SPECIAL WORDS, BUT THE DEFINITION OF WORDS IN A FORTH PROGRAM, MUST BE REVERSE FROM SPECIAL TO GENERAL.

4. FORTH DATA STRUCTURE

SUMMARY

- DATA SIZE
- DATA TYPES
- DATA OPERATIONS
- DICTIONARY ENTRIES
- TABLES AND STRINGS
- SINGLE CHARACTERS

DATA SIZES

BYTE:

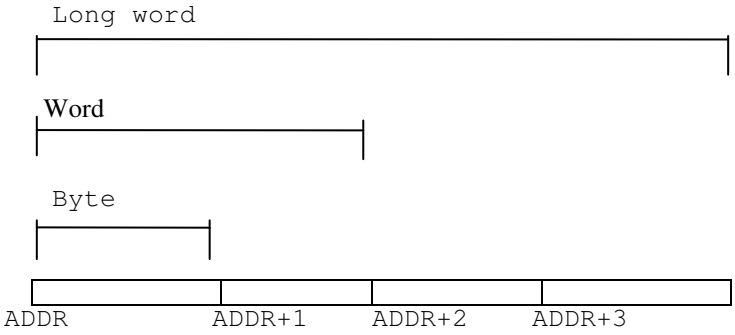
- 8 BIT BYTE OR CHARACTER FORTH WORDS DEALING WITH BYTES NORMALLY BEGIN WITH "C"
  EXAMPLE : C@, C!,

WORD:

- 16 BIT WORD
  FORTH WORDS DEALLING WITH WORDS NORMALLY BEGIN WITH "W"
  EXAMPLE: W@, W!

LONG WORD:

- 32 BITS (STANDARD SIZE, IN THE K1197)



NOTE:
- ADDRESSES ARE 24 BIT ADDRESSES
- ADDRESSES OF WORDS AND LONG WORDS ARE EVEN

DATA TYPES

CONSTANTS:

- FIXED VALUES WHICH ARE PLACED ON THE STACK DIRECTLY BY CALLING THE NAME OF THE **CONSTANT**

VARIABLES:

- VALUES WHOSE ADDRESSED ARE PLACED ON THE STACK BY CALLING THE NAME OF THE VARABLE
- THEY MA BE USED FOR VECTORS, MATRICES BUFFERS ETC.

LITERALS:

- NUMERIC LITERALS, MAXIMUM 0F 32 BITS
  THESE ARE DECIMAL NUMBERS
  EXAMPLES : 123, -15

- EXTENDED NUMERIC LITERALS IN DECIMAL MODE
  PREFIXES ARE:
  - OX FOR HEX
  - OC FOR OCTAL
  - OB FOR BINARY

  EXAMPLES: OX1FFF, 0C177, 0B11111

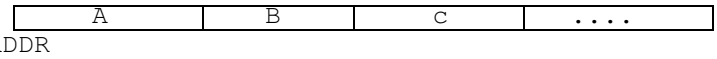
- SHORT IMMEDIATE ASCII LITERALS E.G. #" TEXT" WHERE TEXT MA BE REPLACED BY UP TO 4 CHARACTERS

- LONG POINTER LITERALS
  E.G. " LONG TEXT"
  UP TO 255 CHARACTERS ARE ALLOWED

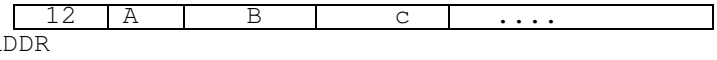
- LONG **HEX** POINTER LITERALS
  E.G. X" O3FF" WHERE HEX CHARACTER PAIRS ARE ENTERED.
  EVEN NUMBER OF CHARACTERS IS NOT TESTED !

STRINGS:

- TYPE 1 MAY BE ACCESSED BY ADDRESS AND EXPLICIT LENGTH
  E.G. ADRS 12 TYPE



- Type 2 (Packed String) May BE ACCESSED BY ADDRESS POINTING TO THE LENGTH BYTE FOLLOWED BY THE STRING



- TYPE 3 SHORT STRING OF MAX. 32 BITS PASSWD ON THE STACK BY #"

NOTE:
PACKED STRING ARE LONG POINTER LITERALS.

DATA OPERATIONS

DEFINING A CONSTANTS:

VALUE **CONSTANT** NAME

EXAMPLE:

```
12 CONSTANT MAX_LENGTH
MAX_LENGTH  .  CR

12 OK
```

NOTE:
CALLING A CONSTANT BY ITS NAME LEAVES THE VALUE OF THE CONSTANT ON TOS

DEFINING A VARIABLE:

INITIAL VALUE **VARIABLE** NAME

EXMAPLE:

```
0 VARIABLE COUNTER
COUNTER . CR

220E12 OK
```

NOTE:
CALLING A VARIABLE BY ITS NAME LEAVES THE ADDRESS OF THE VALUE ON TOS

FETCHING AND STORING VALUES:

@ (pronounce FETCH)

@ (ADDRESS -- CONTENTS OF ADDRESS)

EXAMPLE:

```
COUNTER @ . CR

0 OK
```

! (pronounce STORE)

! (VALUE \ ADDRESS --)

EXAMPLE:

```
1 COUNTER !
COUNTER @ . CR

1 OK
```

DATA OPERATIONS

FETCHING AND STORING VALUES :

+! (pronounce PLUS-STORE)

```
+! ( D\ADDRESS -- )
INCREASES CONTENTS OF ADDRESS BY D
```

EXAMPLE:

```
1 COUNTER +!
COUNTER @ . CR
2 OK
```

NOTE: @, ! AND +! EXPECT EVEN ADDRESSES!
32 BIT VALUES ARE FETCHED AND STORED

FETCHING AND STORING 16 BIT VALUES (WORDS):

W@, W!

USE: LIKE @ AND !

FETCHING AND STORING 8 BIT VALUES (BYTES)

C@, C!

USE: LIKE @ AND !

NOTE: W@, W! NEED EVEN ADDRESSES!

FORTH WORD DP:

THE FORTH WORD DP (DICTIONARY POINTER) POINTS TO THE NEXT AVAILABLE DICTIONARY SPACE.

IT MAY BY ACCESSED BY:
DP @
OR EQUIVALANT BY
HERE

THE DICTIONARY POINTER IS INCREASED BY EACH DEFINING WORD WHICH MAKES A DICTIONARY ENTRY (E.G. CONSTANT, VARAIBLE, :) AND BY **ALLOT** WHICH RESERVES BUFFER SPACE FOR A FIELD.

EXAMPLE:

```
0 VARIABLE BUFF   10 ALLOT   ( 10 bytes will be reserved )
0 VARIABLE BUFF_W  10 ALLOTW  ( 20 bytes will be reserved )
0 VARIABLE BUFF_L  10 ALLOTL  ( 40 bytes will be reserved )
```

Example: Making a table named TABLE[3]

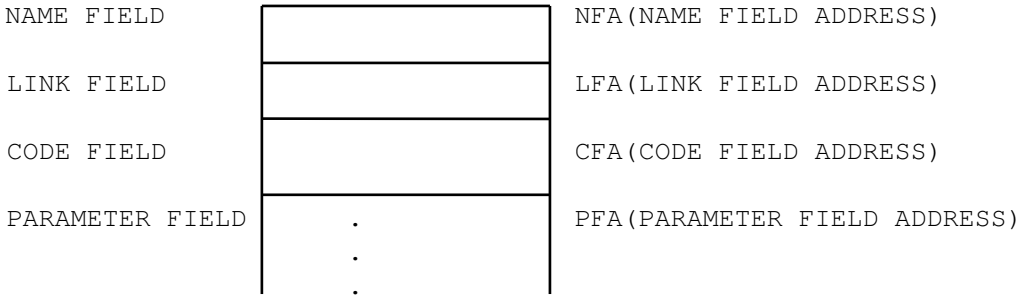
```
1 VARIABLE   TABLE[3]
2 , 3 ,
```

Usage: TABLE[3] 10 DUMP → See the entry for TABLE[3]

DICTIONARY ENTRIES

STRUCTURE OF A DICTIONARY ENTRY:

EACH DICTIONARY ENTRY HAS THE FOLLOWING STRUCTURE:



THE LENGTH OF THE NAME FIELD IS VARIABLE BUT LIMITED TO 31 CHARACTERS.

LINK FIELD AND CODE FIELD HAVE CONSTANT LENGTH FOR EACH DICTIONARY ENTRY (32 BITS EACH)

THE LENGTH OF THE PARAMETER FIELD IS VARIABLE

THE NAME FIELD CONTAINS THE NAME OF THE DICTIONARY ENTRY, E.G.

- NAME OF A VARIABLE
- NAME OF A CONSTANT
- NAME OF A FORTH WORD DEFINED BY " :"

THE LINK FIELD CONTAINS THE NAME FIELD ADDRESS (NFA) OF THE PREVIOUS DICTIONARY ENTRY.

THE CODE FIELD POINTS TO THE CODE OF "WHAT TO DO WITH THE PARAMETER IN THE PARAMETER FIELD"

EXAMPLES:

1) DICTIONARY ENTRY : CONSTANT

THE CFA POINTS TO THE CODE "PUT THE VALUE AT PFA ON THE STACK".
THE VALUE OF THE CONSTANT IS STORED AT PFA AT COMPILE TIME.

2) DICTIONARY ENTRY : VARIABLE

THE CFA POINTS TO THE CODE "PUT THE PFA ON THE STACT".
THE CURRENT VALUE OF THE VARIABLE IS STORED AT PFA.

DICTIONARY ENTRIES

3) LOW LEVEL DEFINITIONS (PRIMITIVES)

THE CFA POINTS TO THE PFA.
THE PARAMETER FIELD CONTAINS EXECUTABLE MACHINE CODE.

4) DICTIONARY ENTRY:

" : " DEFINES A **FORTH WORD** IN TERMS OF OTHER FORTH WORDS ALREADY EXISTENT IN THE DICTIONARY.

THE CFA POINTS TO THE CODE "EXECUTE THE WORDS CORESPONDING TO THE PFA-ENTRIES UNTIL " ; " IS REACHED.

THE PARAMETER FIELD CONTAINS AS MANY ENTRIES AS THERE ARE "OTHER WORDS" INCLUDING THE " ; " .
EACH ENRTY IS THE CFA OF THE CORRESPONDING WORD.

VECTORED EXECUTION

' (PRONOUNCED "TICK") LEAVES THE PFA OF THE FOLLOWING **FORTH WORD** ON TOS

USE: : TEST ." HELLO" ;
' TEST

A **FORTH WORD** WHOSE PFA IS ON TOS CAN BE EXECUTED BY:

CFA EXECUTE

EXAMPLE:
' TEST CFA EXECUTE CR
HELLO OK

CFA CONVERTS THE PFA INTO THE CFA AND **EXECUTE** EXECUTES THE WORD.

THIS IS OFTEN USED, WHEN A TABLE IS FILLED WITH PFA'S OF FORTH WORDS AND A WORD IS EXECUTED ACCORDING TO A CURRENT OFFFSET.

TABLES AND STRINGS

INITIALIZING A TABEL :

WE HAVE SEEN, THAT BUFFER SPACE FOR A TABLE OR AN ARRAY CAN BE RESERVED BY

```
0 VARIABLE BUFF 80 ALLOT
```

BUFF IS INITIALIZED IN ITS 4 LOWEST BYTES WITH ZERO, BUT THE CONTENTS OF THE REST IS STILL UNDEFINED.

```
0 VARIABLE BUFF 1 , 2 , 3 , 4 , 5 ,
```

WILL DEFINE BUFFER SPACE OF 6 LONG WORDS = 24 BYTES AND INITIALIZE THE BUFFER WITH THE VALUES 0 ... 5.

, (pronounce “COMMA”) STORES THE VALUE ON TOS AT HERE AND INCREASES THE DICTIONARY POINTER BY 4.

NOTE:
THE 16 BIT AND 8 BIT VERSIONS OF , ARE W, AND C, .

MOVING CONTENTS OF BUFFER SPACES

CMOVE (ADDR1\ADDR2\N --)

MOVES N BYTES BEGINNING AT ADDR1 TO ADDR2

EXAMPLE:

BEFORE CMOVE

ADDR1 ADDR2 5 CMOVE

AFTER CMOVE

NOTES:

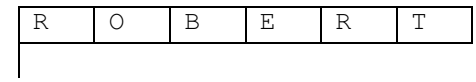
- ADDR1 AND ADDR2 WILL USUALLY BE NAMES OF VARIABLES (EVENTUALLY PLUS AN OFFSET)

- THE 16 BIT VERSION OF **CMOVE** IS MOVE IN THIS CASE ADDR1 AND ADDR2 MUST BE EVEN NUMBERS

TABLES AND STRING

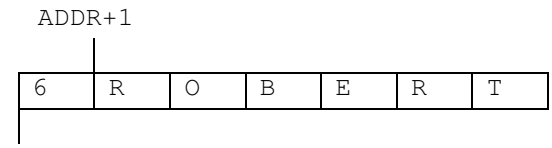
STRINGS

A STRING IS A CHAIN OF CHARACTERS IN BUFFER SPACE.



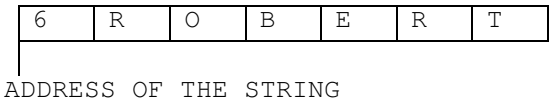
ADDRESS OF THE STRING

A STRING, WHERE THE VALUE OF THE FIRST CHARACTER GIVES THE NUMBER OF CHARACTERS OF THE REST OF THE STRING IS CALLED A PACKED STRING.



IT IS OFTEN USEFUL TO OPERATE WITH PACKED STRING.

THE SEQUENCE " ROBERT" CREATES A PACKED STRING AND LEAVES THE ADDRESS ADDR ON TOS.



ADDRESS OF THE STRING

TO "UNPACK" A PACKED STRING, THERE IS THE FORTH WORD **COUNT**.

COUNT (ADDR -- ADDR+1 \ N)

COUNT INTERPRETS THE VALUE OF THE BYTE AT ADDRESS ADDR AS BYTE COUNT AND LEAVES IT ON TOS.

EXAMPLE:

" ROBERT" COUNT . CR

6 OK

COUNT LEAVES THE NECESSARY STACK PARAMETERS FOR THE STRING OUTPUT COMMAND

- STYPE (CHAR ADDR. \ CHAR COUNT\ --)

EXAMPLES:
" ROBERT" COUNT STYPE CR
ROBERT OK

" ROBERT" 1+ 4 STYPE CR
ROBE OK

STRINGS

STYPE WITH ATTRIBUTES **: STYPE_A**

POSSIBLE ATTRIBUES ARE FORGROUND OR BACKGROUND COLORS AND CHARACTER CODE ACCORDING TO THE FOLLOWING TABLE:

```
FORGROUND            BACKGROUND
WHI_FG = XXXXXX00
                     RED_BG = XXXX11XX
GRN_FG = XXXXXX01
                     BLU_BG = XXXX01XX
YEL_FG = XXXXXX10
                     MAG_BG = XXXX10XX
CYA_FG = XXXXXX11
                     BLK_BG = XXXX11XX

REV_VIDEO = 0001XXXX
ASCII     = 0000XXXX
EBCDIC    = 0010XXXX
HEX SET   = 0100XXXX
TTX       = 0110XXXX
JS8       = 1000XXXX

X = DON'T CARE
0 = CLEARED
1 = SET
```

THE FORMAT OF **STYPE_A** IS :

```
STYPE_A
(CHAR ADDR.\CHAR COUNT\ATTRIBUTE -- )
```

EXAMPLE:

```
" ROBERT" COUNT REG_BG STYPE_A CR
ROBERT OK (RED BACKGROUND)
```

ANY SEQUENCE LIKE

```
" ROBERT"
```

WILL PUT THE STARTING ADDRESS OF THE PACKED STRING ON TOS.
THE ACTUAL STRING IS AUTOMATICALLY STORED IN A SO-CALLED **LITERAL BUFFER**.

THERE ARE 8 ASCII AND 8 HEX LITERAL BUFFERS AVAILABLE IN THE SYSTEM.
THE LITERAL BUFFERS ARE USED "ROUND ROBIN".

THEREFORE, IF A USER NEEDS MORE THAN 8 PACKED STRINGS AT A TIME. HE HAS TO DEFINE HIS OWN BUFFERS (USING VARIABLE AND ALLOT) AND MOVE THE PACKED STRINGS IN THOSE BUFFERS.

EXAMPLE:

```
0 VARIABLE MY-BUFFER 80 ALLOT CR
```

```
" TESTSTRING" DUP C@ 1+
MY-BUFFER SWAP CMOVE CR

MY-BUFFER COUNT STYPE CR

TESTSTRING OK
```

SOMETIMES, A RUNNING PROGRAM EXPEXTS A STRING FROM THE USER BY TERMINAL INPUT.

QUERY IS THE PROPER FORTH WORD TO EXPECT A STRING TERMINATED BY CR AND SAVE IT IN THE TERMINAL INPUT BUFFER (TIB).

THE SEQUENCE c WORD INVESTIGATES TIB FOR A CHARACTER WITH ASCII VALUE c AND STORES THE PACKED STRING DELIMITED BY c AT ADDRESS HERE.

SINGLE CHARACTERS

OUTPUT OF A SINGLE CHARACTER

EMIT (c --)

EMIT DISPLAYS THE ASCII REPRESENTATION OF THE VALUE ON TOS.

EXAMPLE:

```
65 EMIT CR
A OK
```

INPUT OF A SINGLE CHARACTER

KEY (-- c)

KEY EXPECTS THE INPUT OF A SINGLE CHARACTER FROM KEYBOARD; THE ASCII VALUE OF THE CHARACTER IS LEFT ON TOS.

EXAMPLE: (**PAUSE**-FUNCTION)

```
: PAUSE KEY DROP ;
```

PROGRAM EXECUTION STOPS UNTIL A KEY IS PRESSED.
THIS SIMPLE WORD CAN BE USEFUL IN DEBUGGING.

5. FORTH CONTROL STRUCTURES

SUMMARY

- GENERAL
- CONDITIONAL BRANCH
- LOOPS
- LOOPS WITH UNDEINED REPETITION NUMBER
- CASE CONSTRUCT

GENERAL

FORTH SUPPORTS STRUCTURED PROGRAMMING.

THE FOLLOWING CONTROL STRUCTURES ARE STRUCRED PROGRAM BLOCKS.

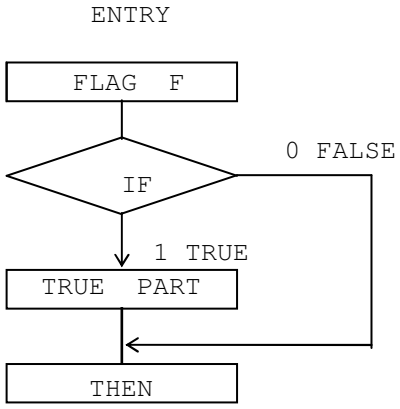
EACH BLOCK HAS ONE ENTRY AND ONE EXIT.

GOTO'S OR JUMPS ARE NOT PERMITTED.

NOTE:
EACH OF THE FOLLOWING CONTROL STRUCTURES IS ONLY ALLOWED WITHIN A FORTH DEFINITION !

CONDITIONAL BRANCH (IF ... THEN)

IF ... THEN CONSTRUCTION



F CAN BE CREATED BY ANY FORTH CODE,
E.G. COMPARISONS

IF TESTS THE FALG

IF F=0 OR “FALSE”, THE PROGRAM
CONTINUES AFTER THEN

IF F <> 0 OR “TRUE”
THE “TRUE PART” IS EXECUTED BEFORE
CONTINUING AFTER THEN

EXIT

IF ... THEN CONSTRUCTIONS

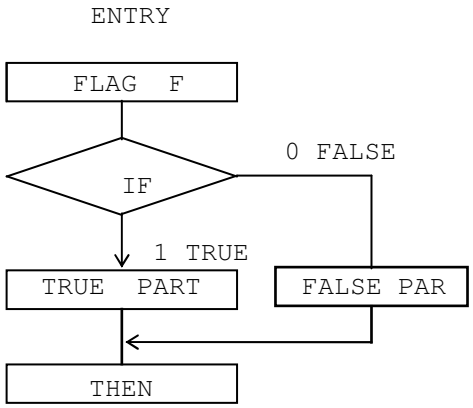
EXAMPLE:

```
( NUMBER -- )

: NEGA 0<                    ( FLAG CREATED )
    IF                       ( TRUE PART )
       ." THIS NUMBER IS NEGATIVE"
    THEN
;
```

NOTES:
- IF CONSUMES THE FLAG
- THE “TRUE PART” MAY CONSIST OF ANY FORTH CODE, BUT EACH CONTROL STRUCTURE CONTAINED IN IT MUST BE CLOSED

CONDITIONAL BRANCH

IF ... ELSE ... THEN CONSTRUCTION



EXAMPLE:

```
( NUMBER -- )

: TEST1 0<      ( FLAG CREATED )
   IF           ( TRUE PART)
      ." THIS NUMBER IS NEGATIVE"
   ELSE
      ." THIS NUMBER IS NONNEGATIVE"
   THEN
;
```

NOTES:
- THE PREVIOUS NOTES APPLY TOO
- THE "FALSE PART" MAY ALSO CONTAIN ONLY CLOSED CONTROL STRUCTURES
- ELSE SEPARATES THE "TRUE PART" FROM THE "FALSE PART"

THE **IF ELSE THEN** CONSTRUCTION WORKS LIKE **IF THEN** EXCEPT THAT IN F=0 THE "FALSE PART" IS EXECUTED BEFORE CONTINUING AFTER THEN.

NESTING OF IF ... ELSE ... THEN CONSTRUCTION

THE IF ... THEN ... AND IF ... ELSE ... THEN STRUCTURES MAY BE NESTED IN NEARLY ARBITRARY DEPTH.

EXAMPLE OF NESTING:

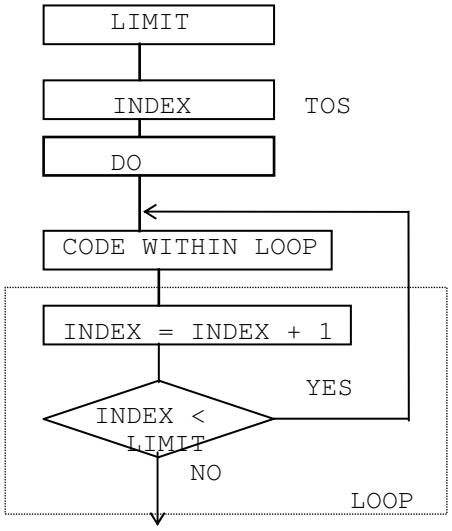
```
( NUMBER -- )
: TEST2
  DUP 0 >                    ( FLAG CREATED)
  IF
       ." POSITIVE "
       5 >                   ( NEW FLAG CREATED )
       IF
          ." AND GREATER THAN 5"
       THEN
  ELSE
       0=
       IF                    ( NEW FLAG CREATED )
         ." EQUAL TO ZERO"
       ELSE
          ." NEGATIVE"
       THEN
  THEN
```

LOOPS

DO ... LOOP
DO ... +LOOP (INDUCTIVE LOOPS)

USED FOR REPETITION OF THE CODE BETWEEN **DO** AND **LOOP** OR **+LOOP** RESPECTIVELY. THERE IS A LOOP COUNTER, WHICH MUST BE GIVEN A STARTING (INDEX) AND TERMINATING (LIMIT) VALUE.

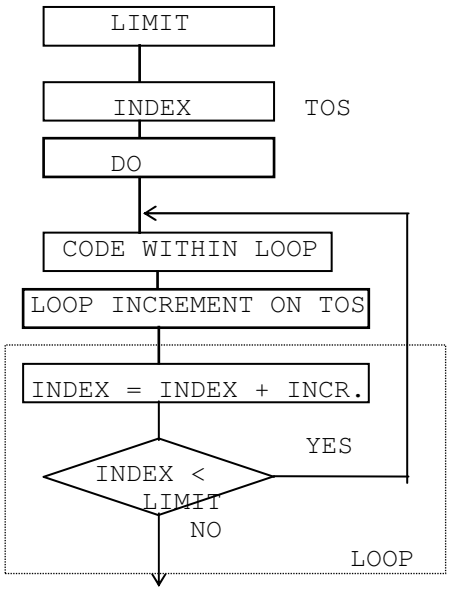
DO ... LOOP




```
EXAMPLE:

0 VARIABLE VAR

: TEST3 ( -- )
    10              ( LIMIT )
    0               ( INDEX )
    DO
       VAR @ .      ( OUTPUT VAR )
       3 VAR +!     ( INCREASE VAR BY 3 )
    LOOP
;
```

USE:
```
TEST3 CR
0 3 6 9 12 15  18 21 24 27 OK
```

DO ... +LOOP




```
EXAMPLE:

0 VARIABLE VAS

: TEST4 ( -- )
    30            ( LIMIT )
    0             ( INDEX )
    DO
      VAS @ .   ( OUTPUT VAS )
      3 VAS +!  ( INCREASE VAS BY 3 )
      3         ( LOOP COUNTER INCREMENT)
    +LOOP
 ;
```

USE:
```
TEST4 CR
0 3 6 9 12 15  18 21 24 27 OK
```

LOOPS

DO ... LOOP
DO ... +LOOP (INDUCTIVE LOOPS)

NOTES:

- THE FORTH WORDS WITHIN THE LOOP ARE EXECUTED AT LEAST ONCE SINCE THE TESTING OF THE TERMINATING CONDITION OCCURES AT THE END OF THE LOOP
- DO MOVES INDEX AND LIMIT FORM THE PARAMETER STACK TO THE RETURN STACK
- THE CURRENT VALUE OF THE LOOP COUNTER CAN BE ACCESSED BY THE **FORTH WORD** I WHICH LEAVES THE VALUE ON TOS
- IF THE LOOP INCREMENT IS POSITIVE, THE LOOP IS LEFT WHEN I IS **GREATER THAN** OR EQUAL TO THE TERMINATING VALUE. IF THE INCREMENT IS NEGATIVE, THE LOOP IS LEFT WHEN I IS SMALLER THAN THE TERMINATING VALUE.
- THE CODE WITHIN A LOOP MAY CONTAIN ONLY CLOSED CONTROL STRUCTURE

NESTING OF LOOPS

- LOOPS MAY BE NESTED IN NEARLY ARBITRARY DEPTH
- THE LOOP COUNTER OF THE ACTUAL LOOP CAN BE ACCESSED BY **I**
- THE LOOP COUNTER OF THE NEXT OUTER LOOP CAN BE ACCESSED BY **J**
- FURTHER LOOP COUNTERS MUST BE ADMINISTERED BY THE PROGRAMMER HIMSELF

EXAMPLE:

```
:IJ_LOOP
   4 0
   DO
        ." I = " I . CR
        4 0
        DO
            ." I =' I . SPACE ." J =" J . CR
        LOOP
    LOOP
;
```

LEAVING A DO ... LOOP OR DO ... +LOOP

THE ONLY WAY TO LEAVE ONE OF THE ABOVE LOOPS IS TO REACH THE LIMIT VALUE.
THE **FORTH WORD LEAVE**, WHEN EXECUTED BETWEEN DO AND LOOP OR +LOOP CHANGES THE STORED LIMIT VALUE TO THE CURRENT VALUE OF THE LOOP INDEX.
IN THIS CASE THE LOOP IS LEFT WHEN LOOP OR +LOOP IS REACHED NEXT TIME.

EXAMPLE:

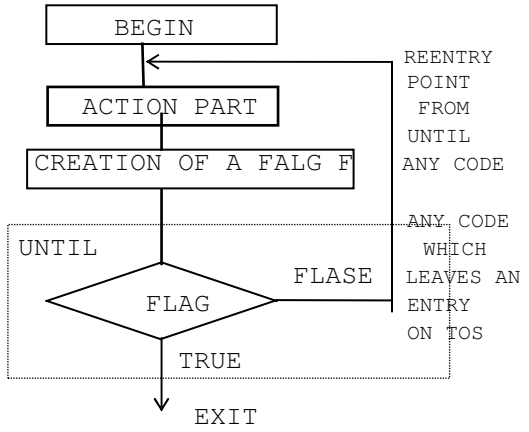
```
: TEST5
  10 0
  DO
    I DUP . 5 >  IF
       LEAVE
    THEN
   LOOP
;
```

USE :

```
TEST5 CR
0 1 2 3 4 5 6 OK
```

LOOPS WITH UNDEFINED REPETITION NUMBER

BEGIN ... UNTIL



EXAMPLE:

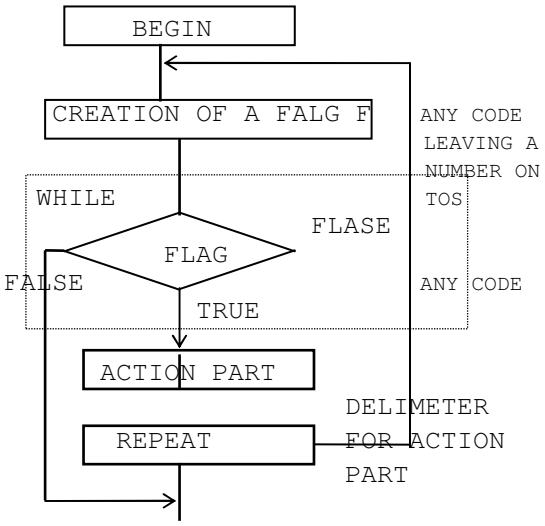
```
: TEST6 ( -- )

  BEGIN
     ." PRESS Y KEY: "  (ACTION PART)
     KEY
     89 =          ( CREATION OF FLAG)
  UNTIL
  ." END"
;
```

NOTE:
- UNTIL CONSUMES THE FLAG
- POSIBLE CONTROL STRUTURES IN THE ACTION PART MUST BE CLOSED

BEGIN ... AGAIN

- THIS IS AN INFINITE LOOP.
- AGAIN CAUSES AN UNCONDITIONAL JUMP TO THE CODE AGTER BEGIN
- THERE IS NO SPECIAL FORTH WORD TO LEAVE SUCH A LOOP, BUT **EXIT** WHEN EXECUTED BETWEEN **BEGIN** AND **AGAIN** MAY SERVE FOR THIS PURPOSE.
- " EXIT" LEAVES THE DEFINITION IN WHICH IT IS CALLED AND RETURNS TO THE DEFINITION WHICH CALLED IT OR TO THE OUTER INTERPRETER.

BEGIN ... WHILE ... REPEAT



EXAMPLE:

```
: TEST7 ( -- )

  BEGIN
     ." PRESS Y KEY: "
     KEY
     89 -          ( CREATION OF FLAG)
  WHILE
     ." WRONG CHOICE" CR ( ACTION PART)
  REPEAT
     ." RIGHT CHOICE " CR
```

NOTES:

- WHILE CONSUMES THE TOS
- POSSIBLE CONTROL STRUCTURES IN THE ACTION PART MUST BE CLOSED
- UNLIKE DO ... LOOP OR BEGIN ... UNTIL THE TERMINATION CONDITION IS TESTED BEFORE THE ACTION PART

CASE CONSTRUCT

THE CASE CONSTRUCT PROVIDES A MECHANISM FOR SELECTING ONE OF A SET OF POSSIBLE CASES.
THE FOLLOWING EXAMPLE SHOWS THE USAGE OF THE RELATED WORDS:

```
( VALUE -- )
: TEST_CASE
  DOCASE
    CASE 1 { ." VALUE = 1" }
    CASE 2 { ." VALUE = 2" }
    CASE 3
    ORCASE 4 { ." VALUE = 3 OR 4" }
    CASE DUP { ." OTHER VALUE" }
  ENDCASE
;
```

NOTES:

- IF EQUALLITY IS FOUND, THE { } - ACTION IS PERFORMED AND THE PROGRAM EXECUTION CONTINUES AGTER ENDCASE; NO FURTHER COMPARISONS ARE EXECUTED
- CASE DUP EXPRESSES THE DEFAULT CONDITION AND MUST BE PLACED AS LAST CASE

6. WORD LIST

WORD	STACK NOTATION	MODULE
!	(D \ ADDR ---) STORES 32 BITS OF D AT ADDR	DA
#>>	SHIFTS 32 BIT VALUE BY N BITS TO THE RIGHT USE: N #>>	
( )	COMMENT	
*****	(N1 \ N2 -- PROD) MULTIPLICATION	FC
***/**	(N1\N2\N3 -- N1*N2\N3) LEAVES QUOTIENT	
+	(N1\N2 -- SUM) ADDITION	
+!	(D\ADDR --) INCREASES CONTENTS OF ADDR BY D	
+LOOP	LIKE LOOP, BUT THE INDEX IS CHANGED BY THE VALUE ON TOS	CO
-	(N1\N2 -- DIFF) SUBTRACTION	
.	OUTPUTS TOS ENTRY AS VALUE ACCORDING TO BASE; REMOVES TOS ENTRY	FC
."	COMPILER WORD FOR STRING OUTPUT	DE
.S	OUTPUTS TOP 4 STACK ENTRIES DOES NOT REMOVE THE ENTRIES	FC
/	(N1\N2 -- QUOTIENT) DIVISION	
/MOD	(N1\N2 -- REMAINDER\QUOTIENT)	
:	DEFINING WORD FOR A FORTH WORD	DE
;	TERMINATES DEFINITION OF A FORTH WORD	DE
<	(N1\N2 -- FLAG) TRUE IF N1 < N2	
<<#	SHIFTS 32 BIT VALUE BY N BITS TO THE LEFT USE: N <<#	
=	(N1\N2 -- FLAG) TRUE IF N1 = N2	
>	(N1\N2 -- FLAG) TRUE IF N1 > N2	

WORD	STACK NOTATION	MODULE
>R	(N --) MOVE VALUE FROM PARAMETER STACK TO RETURN STACK	
?DUP	(N1 -- N1\N1) IF N1 <> 0 (N1 -- N1) IF N1 = 0	
@	(ADDR -- VALUE) FETCHED 32 BIT VALUE FROM ADDR	DA
0<	(N -- FALG) TRUE IF N < 0	
0=	(N -- FLAG) TRUE IF N = 0	
2DROP	(N1\N2 --)	
2DUP	(N1\N2 -- N1\N2\N1\N2)	
3DROP	(N1\N2\N3 --)	
3DUP	(N1\N2\N3 -- N1\N2\N3\N1\N1\N2\N3)	
ABS	(N -- |N|) LEAVESJ ABSOLUTE VALUE	
AGAIN	TERMINATES REPETITIVE SEQUENCE	CO
ALLOT	(N --) RESERVES N BYTES OF DICTIONARY SPACE	DA
ALLOTL	(N --) RESRVES N LONG WORDS OF DICTIONALRY SPACE	DA
ALLOTW	(N --) RESERVES N WORDS OF DICTIONARY SPACE	
BEGIN	STARTS REPETITIVE SEQUENCE	CO
C!	(C\ADDR --) STORES 8 BITS OF C AT ADDR	DA
C@	(ADDR -- VALUE) FETCHES 8 BIT VALUE FROM ADDR	
CASE	PROVIDES MECHANISM FOR SELECTING ONE OF A SET OF POSSIBLE CASES	
CFA	(PFA -- CFA)	DA (CODE FIELD ADDR)
CMOVE	(ADDR1\ADDR2\N --) MOVES N BYTES BEGINNING AT ADDR1 RO ADDR2	DA
CONSTANT	DEFINING WORD FOR A CONSTANT	DA
COUNT	(ADDR -- ADDR+1\N)	DA
CR	NEXT OUTPUT IN NEW LINE	

WORD	STACK NOTATION	MODULE
DO	(LIMIT \INDEX --) STARTS DO LOOP	CO
DP	DICTIONARY POINTER	DA
DROP	(N --)	FC
DUP	(N1 -- N1\N1)	
ELSE	SEPARATES “TRUE PART” FROM “FALSE PART”	CO
EMIT	(C --)	DA
ENDIF	SYNONYM TO THEN	CO
EXECUTE	(CFA --) EXECUTES FORTH WORD WITH CODE FIELD ADDRESS CFA	
EXIT	LEAVE DEFINITION	
FORGET	EXECUTED IN THE FORM: FORGET CCCC DELETES DEFINITION NAME CCCC FROM THE DICTIONARY WITH ALL ENTRIES PHYSICALLY FOLLOWING IT (BEGIN ABOVE CCCC)	

FILL	**(addr/quan/b --)** Fills memory at the address with the specified quantity of bytes b. “quan” is a 16-bit value, i.e. a maximum of 65536 (2^{16}) bytes can be filled. 0 VARIABLE AAA 10 ALLOT CCC 3 1 FILL → CCC의 3 bytes가 1 로 채워진다. 참조: FILLW
HERE	SAVE AS DP @

I	(-- LOOP INDEX) INNERMOST LOOP	CO
IF	(FLAG --)	CO
J	(-- LOOP INDEX) NEXT OUTER LOOP	CO
KEY	(-- C)	DA
LEAVE	EARLY TERMINATION OF A LOOP	CO
LOOP	TERMINATES REPETITIVE SEQUENCE	CO
MAX	(N1\N2 -- MAXIMUM VALUE OF N1, N2)	
MIN	(N1\N2 -- MINIMUN VALUE OF N1, N2)	
MOD	(N1\N2 -- REMAINDER OF N1\N2)	
MOVE	(ADDR1\ADDR2\N --) MOVES N WORDS BEGINNING AT ADDR1 TO ADDR2	
OVER	(N1\N2 -- N1\N2\N1)	FC
PICK	(N -- M) COPY OF Nth STACK ENTRY ON TOS	FC
QUERY	EXPECTS 80 CHARACTERS OF TEXT (OR UNTIL CR) AND SAVE IT TO TIB	DA

WORD	STACK NOTATION	MODULE
R	(-- N) COPIES VALUE FROM RETURN STACK TO PARAMETER STACK	
R>	(-- N) MOVES VALUE FROM RETURN STACK TO PARAMETER STACK	
REPEAT	TERMINATES REPETITIVE SEQUENCE	CO
ROT	(N1\N2\N3 -- N2\N3\N1)	FC
S.	OUTPUTS TOS ENTRY AS VALUE ACCORDING TO BASE; DOES NOT REMOVE TOS ENTRY	FC
SPACE	NEXT OUTPUT SHIFTED BY ONE SPACE	
SPACES	(N --) NEXT OUTPUT SHIFTED BY N SPACES	DE
STACK	OUTPUTS COMPLETE STACK WITHOUT REMOVING TE VALUES	
STYPE	(CHAR ADDRESS\CHAR COUNT --) OUTPUT A STRING	DA
STYPE_A	(CHAR ADDRESS\CHAR COUNT\ATTR --) STYPE WITH ATTRIBUTE	DA
SWAP	(N1\N2 -- N2\N1)	FC
THEN	DELIMITING WORD FOR CONDITIONED EXECUTION	CO
UNTIL	(FLAG --) TERMINATES REPETITIVE SEQUENCE	CO
VARIABLE	DEFINING WORD FOR A VARIABLE	DA
VLIST	LISTS THE NAMES OF THE DEFINITIONS IN THE DICTIONARY	
W!	(N\ADDR --) STORES 16 BITS OF N AT ADDR	
W@	(ADDR -- VALUE) FETCHES 16 BIT VALUE FROM ADDR	DA
WHILE	(FLAG --) KEYWORD FOR CONDITIONALLY REPETITION	
WORD	(C --) READS THE NEXT CAHR. FROM INPUT STREAM UNTIL DELIMETER WITH ACSII VALUE C IS FOUND	

DIA Communications, Inc.

이교순 / 011-704-1297

E-Mail: kyosoon@diacomm.com