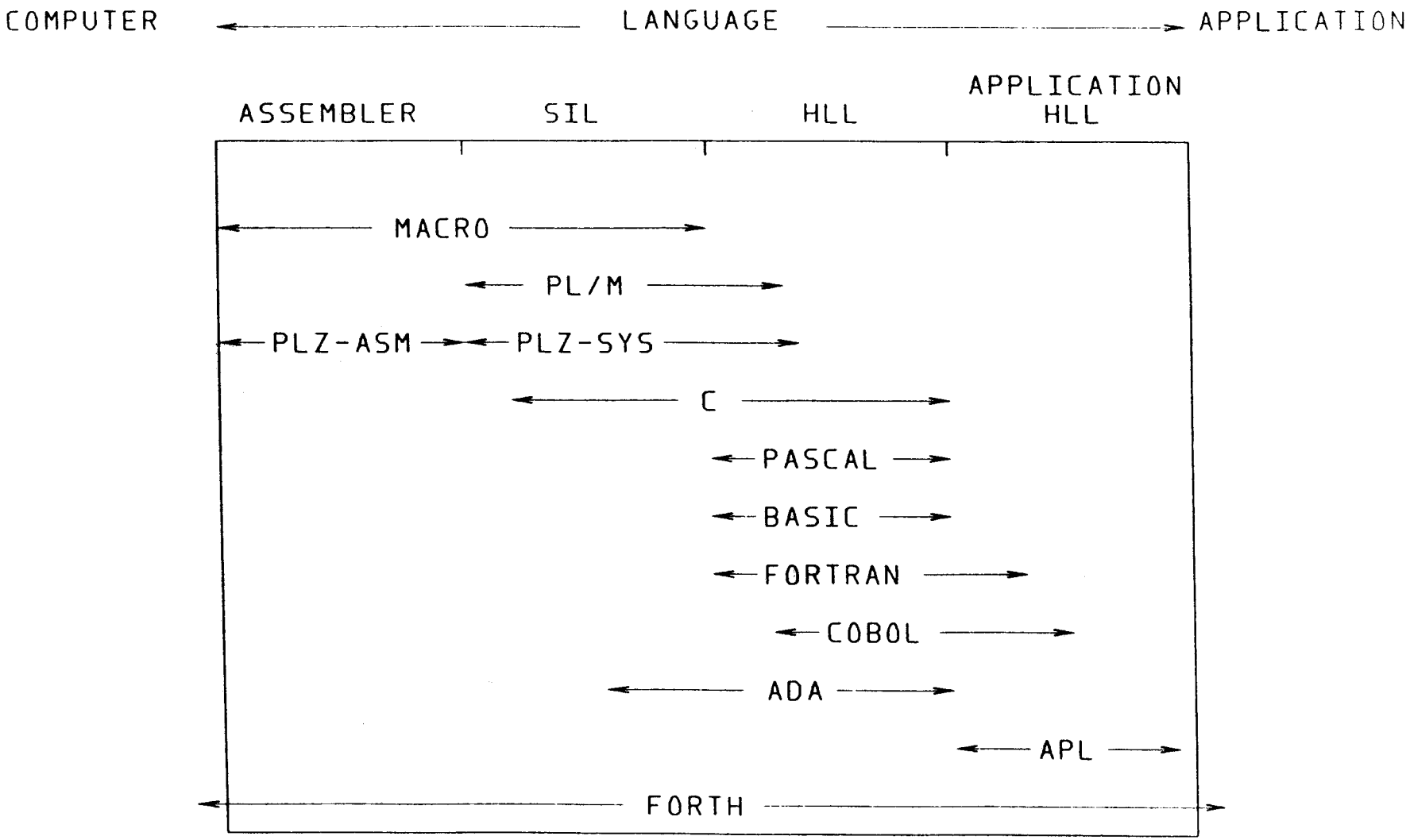
FORTH SEMINAR VIEWGRAPHS

After teaching FORTH for some time, a course structure was shaped up as an eight session sequence, treating FORTH from the introduction of stacks to the inner interpreters and to the programming style. I have kept the notes as viewgraphs on disks in the form of FORTH screens. The 16 by 64 format of FORTH screens shows very well on viewgraphs if the text is double spaced. Studies showed that a viewgraph or a slide should not contain more than six distinct ideas on the same graph. FORTH screens can be arranged very comfortably this way. Although I am printing these screens three on one page, they are ideally used one screen per viewgraph.

It is said that a small duckling follows the first moving being it sees as its mother. In the same way I was influenced by Kim Harris who gave me my first FORTH lectures. Many screens in the introduction part are from his notes. It will not be suprising if many of you find them familiar. Much material in the last session on FORTH style was adopted from his paper in the 1980 FORML Procedings. In the middle session on the internal structures of the FORTH computer I used the Systems Guide to fig-FORTH as reading assignments. Many of the figures therein are useful in the presentations.

The sessions were designed with heavy orientation towards professional engineers and programmers. The language structure of FORTH and the mechanisms in the virtual FORTH machine are emphasized. I saw quite a few blank looking eyes and a number of sound sleepers in my audiences. I hope that these symptoms of indigestion were only temporary and the materials somehow would become obvious when they had to get into the codes in a FORTH system and make use of them.

By presenting these screens in the form that can be copied directly onto viewgraph transparencies, I hope that those of you who may have to get up and explain FORTH to other people have a reference point from which to start. The selection, ommision, modification, or addition to these materials will be determined by your style and preference, and also on the needs of your audience.



7. Comparison of Computer Languages

SESSION I. ADVANTAGES OF FORTH

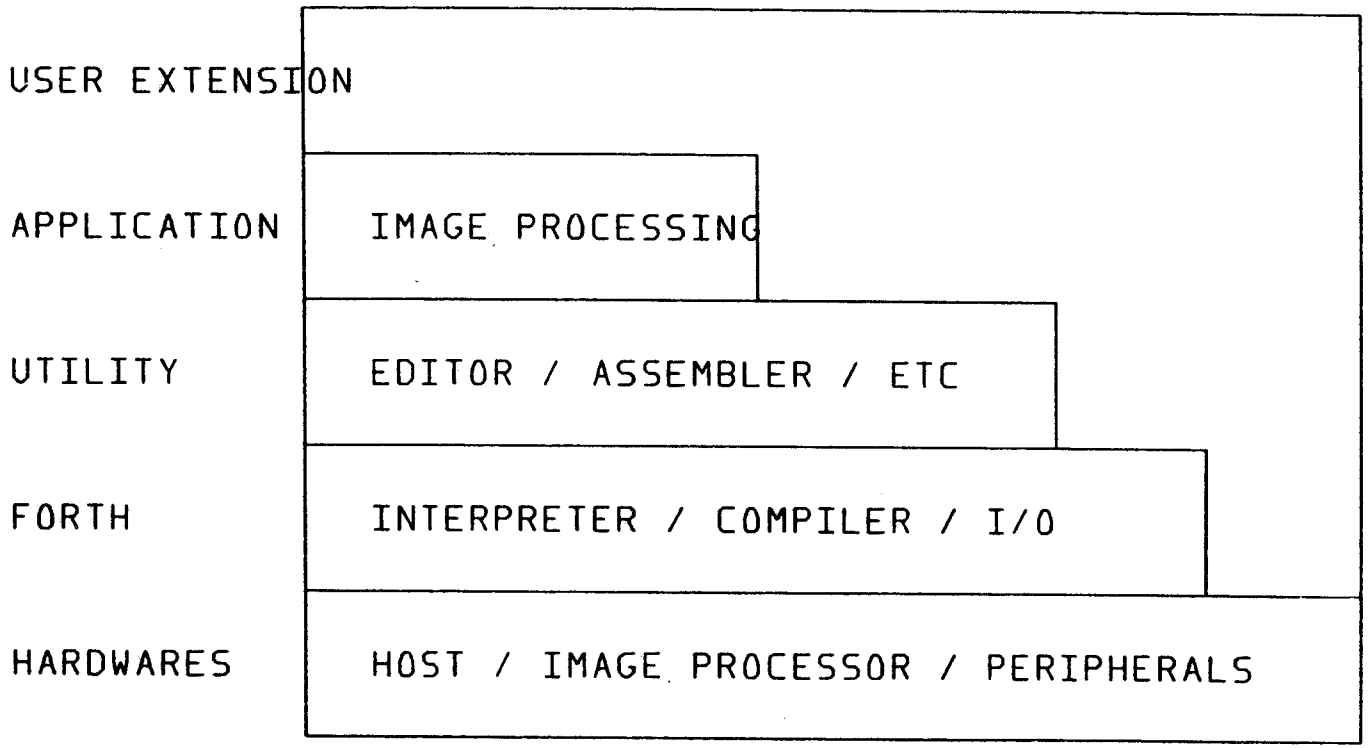
- HIGH SOFTWARE QUALITY
- LOW PROGRAMMING COSTS
- SHORT PROGRAM DEVELOPMENT TIME
- COMPACT AND ROMMABLE CODES
- INTERACTIVE ENVIROMENT
- USER PRGRAMMABILITY

SOFTWARE QUALITY

- PROOF OF PROGRAM CORRECTNESS
- STRUCTUREDNESS
- MODULARITY
- COMPACTNESS

EXTENSIBILITY OF THE SYSTEM

- USER DEFINED INSTRUCTIONS
- NEW SYNTAX AND DATA STRUCTURES (COMPILER/INTERPRETERS)
- SELF REGENERATION (TARGET COMPILATION)



8. Extensions of FORTH Computer

INTERACTIVE OPERATIONS

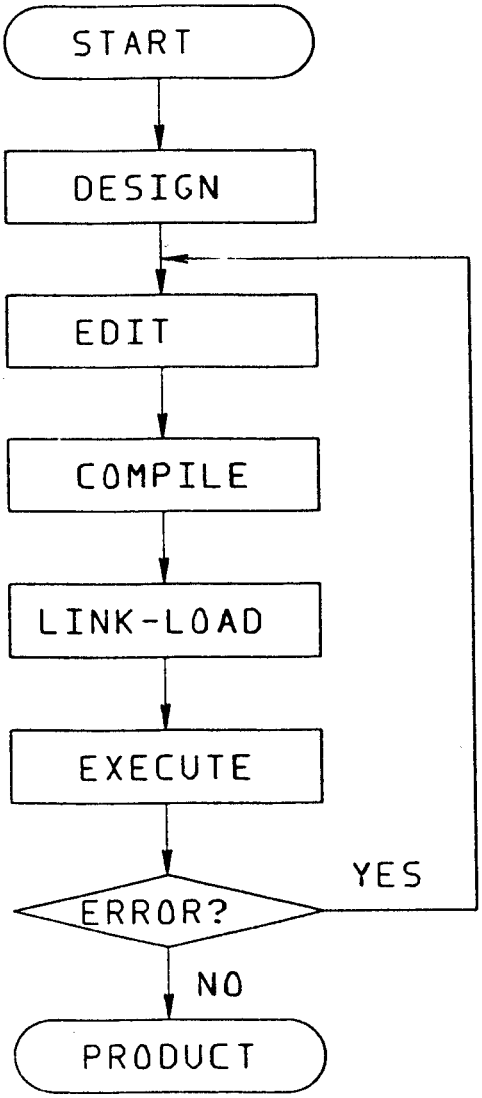
- ALL FUNCTIONS DIRECTLY INVOKED BY ASCII NAMES
- IMMEDIATE COMPILATION AND TESTING
- DIRECT CONTROL OVER COMPUTER RESOURCES
- DIRECT ACCESS TO MEMORY AND I/O DEVICES
- FAST COMPILATION AND INTERPRETATION

WHAT IS FORTH ?

- A PROGRAMMING LANGUAGE
- AN OPERATING SYSTEM
- A STRUCTURED ASSEMBLY LANGUAGE
- A RELIGION AND A CULT
- AN UNREDEEMABLE ADDICTON

CHARACTERISTICS OF FORTH

- DUAL STACK ARCHITECURE
- MEMORY RESIDENT DICTIONARY
- COMPILER-INTERPRETER
- VIRTUAL MEMORY
- INDIRECT THREADED CODES



CYCLE TIME:

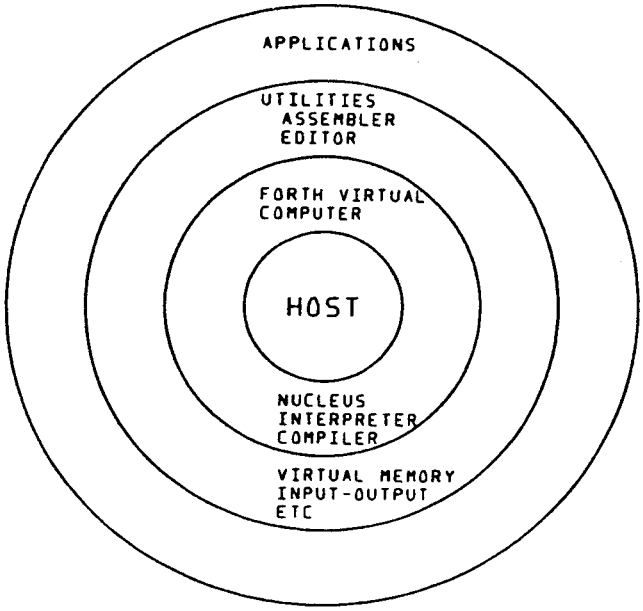
HLL - 20 MINS

FORTH - 10 SECS

9. Program Development Cycle

INSTRUCTION SET OF FORTH

- STACK MANIPULATION
- ARITHMETIC AND LOGIC OPERATIONS
- COMPILER AND STRUCTURE OPERATIONS
- TERMINAL AND DISK I/O OPERATIONS
- MEMORY AND DICTIONARY OPERATIONS
- MISCELLANEOUS OPERATIONS



10. Layered Structure of FORTH Computer

SESSION II. INTRODUCTION TO FORTH

OBJECTIVE
Learn to use FORTH to write simple programs and to test them interactively.

TOPICS
. FORTH instruction set
. Data stack and return stack
. Numbers and data representations
. Execute FORTH instructions
. Compile new instructions
. Write simple FORTH programs
. Terminal input and output

FORTH LANGUAGE

WORD

A sound or a combination of sounds, or its representation in writing, that symbolizes and communicates a meaning...

A command or an order.

(from the American Heritage Dictionary.)

FORTH SYNTAX

PROGRAM

A sequence of FORTH words, separated by spaces, or possibly by a CR (carriage return).

WORD

A sequence of ASCII characters except spaces, CR's, or BS (Back Spaces which erases the previous character).

UNIQUENESS

Words must be unique by their first 3 (or 31) characters and the length of characters.

EXAMPLES OF FORTH WORDS

FORTH words must be either instructions or numbers.

Instructions:

FORTH DUP DROP AND XOR TYPE @ ! +! * / + - MOD

Numbers:

1 2 3 -1 -2 -3 12345 123.45

STACK USAGE

Numbers entered are pushed onto a data stack and removed in the last-in first-out (LIFO) order.

```
1 2 3   ok
. 1 ok
. 2 ok
. 3 ok
. 0 STACK EMPTY
```

ARITHMETICS

The stack usage leads to the postfix order of operands and operators.

Postfix notation = Reverse Polish Notation (RPN)

```
1 2 + .   3 ok
7 2 * 1 + .  15 ok
3 2 - .   1 ok
```

NOTATIONAL CONVENTIONS

```
Word   ( input stack parameters --- output stack parameters )

         top of stack listed last --->
```

Abbreviations of data types:

Flag	f	(true..false)
ASCII character	c	(0..127)
Byte	b	(0..255)
Unsigned integer	un	(0..65355)
Signed integer	n	(-32768..32767)
Address	addr	(0.. 65355)
Double integer	d	(-2147483648..2147483647)
Unsigned double integer	ud	(0..4294967295)

ARITHMETIC OPERATORS

```
0 -1 MIN .    -1 ok
17 4 MAX .    17 ok
-3 ABS .    3 ok
3 NEGATE .    -3 ok
```

The composite operators */ and */MOD are useful for scaled, fixed point calculations:

```
20000 5 100 */ .    1000 ok          (5% of 20000)
20000 55 1000 */ .    1100 ok        (5.5% of 20000)
```

32 BIT DOUBLE INTEGERS

Each double integer takes 2 stack cells, the high order 16 bits are on the top of the stack and the low order 16 bits below.

```
12.3 D.    123 ok
1.23 D.    123 ok
123. D.    123 ok
123. . .    0 123 ok
123 0 D.    123 ok
```

```
DROP        2
            1       -->         1

DUP                             1
            1       -->         1

SWAP        2                   1
            1       -->         2

OVER                            1
            2       -->         2
            1                   1

ROT         3                   1
            2       -->         3
            1                   2
```

EXAMPLES OF STACK INSTRUCTIONS

```
3 DUP * .   9 ok
5 DUP * .   25 ok

: SQUARE   DUP * ;   ok

3 SQUARE .   9 ok
5 SQUARE .   25 ok
```

SESSION III. PROGRAMMING IN FORTH

OBJECTIVE

Learn the general procedures in writing FORTH programs.

TOPICS

. Constants
. Variables
. Colon Definitions
. Numbers and Arrays
. Strings

SYMBOLIC CONSTANTS

Defining a constant:

```
number   CONSTANT   name
```

Examples

Definitions	Usage
10 CONSTANT TEN	TEN . 10 ok
9430 CONSTANT MY-ZIP	MY-ZIP . 9430 ok

```
MY-ZIP TEN 3 * + .   9460 ok
```

SYMBOLIC VARIABLES --- A symbol whose value can be changed.

Defining a variable

```
         VARIABLE   name
```

Examples

```
VARIABLE  X
VARIABLE  ZIP
```

Instructions operating on variables:

```
variable_name   @              ( fetch the value )
new_value variable_name   !     ( change the value )
```

HOW VARIABLES WORKS

variable_name --- addr

```
BASE .   10294 ok
X .   7920 ok
ZIP .   7930 ok

BASE . 10294 ok
10294 @ .   10 ok
8 10294 !   ok
TEN .   12 ok
MY-ZIP ZIP !   ok
ZIP @ .   22326 ok
```

BASE CONVERSION OF NUMBERS

Example:

```
16 HEX .   10 ok
7FFF DECIMAL .   32767 ok
40 3 * 7 + DUP . HEX .   127 7F ok
```

The conversion is controlled by the contents of variable BASE.

```
: HEX   16 BASE ! ;   ok
: DECIMAL 10 BASE ! ;   ok
: OCTAL   8 BASE ! ;   ok
: BINARY   2 BASE ! ;   ok
TEN OCTAL .   12 ok
BINARY 100111 OCTAL .   47 ok
```

INSTRUCTIONS AND DICTIONARY

FORTH Instructions: Named, linked, and executable routines stored in the memory of a FORTH computer.

FORTH Dictionary: The entire linked list of FORTH instructions in the memory of a FORTH computer.

DEFINING NEW INSTRUCTIONS

```
:   new_instruction   list_of_previously_defined_words   ;
```

Examples:

```
: 8*   2* 2* 2* ;   ok
: %   100 */ ;   ok

7 8* .   56 ok
200 5 % .   10 ok
```

ADDRESS MANIPULATIONS

Define a variable array:

```
VARIABLE TABLE   6 ALLOT   ( Array size 4 cells)

1 TABLE !                  ( Array initializing)
2 TABLE 2 + !
3 TABLE 4 + !
4 TABLE 6 + !

TABLE @ .   1 ok
TABLE 4 + @ .   3 ok
```

FORTH INSTRUCTIONS

Standard Instructions: Instructions provided by a FORTH system. They are invoked by their names.

User Instructions: Instructions defined by the user. They must first be defined before being invoked.
User instructions include high level colon instructions, low level machine code instructions, constants, variables, vocabularies, etc.

NUMBERS IN FORTH

Number: A sequence of digits delimited by spaces or CR's. Digits are ASCII characters starting from 0. The total number of digits is determined by the contents of the variable BASE. If the contents of BASE is greater than 10, ASCII characters starting at A are included in sequence.

ADDRESS MANIPULATIONS: PSEUDO VARIABLE ARRAYS

Defining a variable array:

```
VARIABLE  TABLE  6  ALLOT    ( Size 4 cells)
```

Initializing the array:

```
1 TABLE  !       ( 1st cell)
2 TABLE 2 + !    ( 2nd cell)
3 TABLE 4 + !    ( 3rd cell)
4 TABLE 6 + !    ( 4th cell)
```

Accessing cells in the array:

```
TABLE  @  .    1 ok
TABLE  4 + @ .   3 ok
```

ACCESSING AN ARRAY

To simplify cell selection and to improve readability, define:

```
: ()    ( subscript array --- addr )    SWAP 2* + ;
```

Accessing the array elements:

```
0 TABLE () @ .    1 ok
2 TABLE () @ .    3 ok
```

CREATE AN INITIALIZED VARIABLE ARRAY

```
VARIABLE TABLE   2  ,  3  ,  4  ,   1 TABLE !
```

Accessing the array:

```
2 TABLE () @ .   2 ok
-15   2 TABLE () !   ok
TABLE 2 + ?   -15 ok
```

STRING INSTRUCTIONS

String instruction is followed immediately by the string used by the instruction. The string instruction may specify a character other than space as the delimiter of the string.

Comment: (xxxx yyyy zzzz)
'(' is the comment instruction and ')' is the delimiter.

String output: ." xxxx yyyy zzzz"
'."' is the string instruction with '"' as its delimiter.

OTHER STRING INSTRUCTIONS

Defining instruction use the following string as the name of the defined user instruction. Examples are : , CODE , CONSTANT VARIABLE , and VOCABULARY .

Editor instructions use strings for text entry and modification.

SESSION IV. STRUCTURED PROGRAMMING

Successive Refinement: Hierarchical decompositiion of a problem into smaller parts.

Modular structures: Each part or module has only one entry point and only one exit point.

STRUCTURES

Sequential Operations:

```
step 1 --- step 2 --- step 3 ---  ...
```

Selection:

```
--- test ---  true part ---  ...
          ---  false part ---
```

Loop:

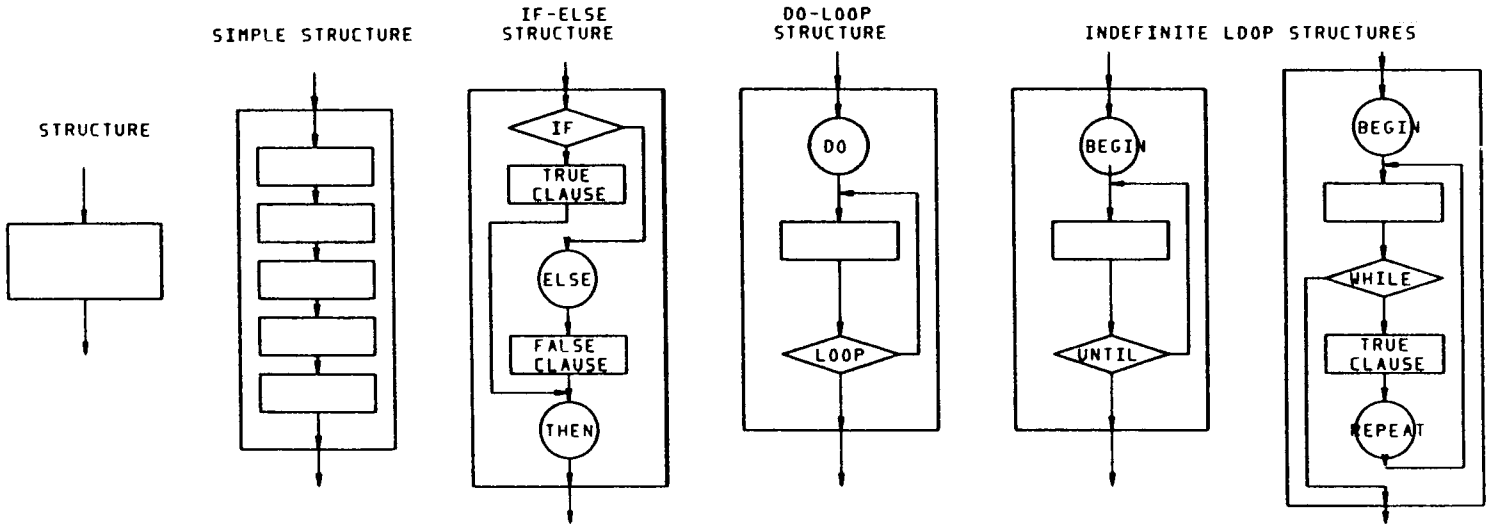
```
--- step 1 --- ... --- step n --- test ---
   <-----------------------------------
```

FORTH CONTROL STRUCTURE FOR SELECTION

```
value   IF   true_part   ELSE   false part   THEN
```

The stack value=0 meams false; value<>0 means true.

```
: TEST   IF ." TRUE"   ELSE ." FALSE"   THEN ;

1 TEST   TRUE ok
0 TEST   FALSE ok
-15 TEST   TRUE ok
```

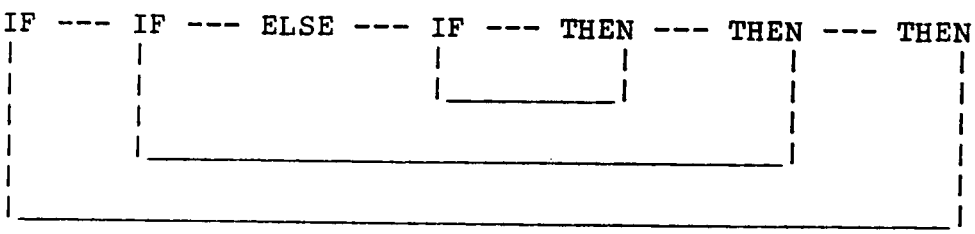


11. Structures in FORTH Language

COMPARISON OPERATORS

```
0 0= TEST    TRUE ok
1 0= TEST    FALSE ok
-1 0< TEST    TRUE ok
5 0< TEST NOT    TRUE ok

4 3 = TEST    FALSE ok
-4 -3 < TEST    TRUE ok
1 10 > TEST    FALSE ok

0.0 D0= TEST    TRUE ok
5 0 D0= TEST    FALSE ok
-2000.000 -1999.999 D< TEST    TRUE ok
```

NESTED IF STRUCTURES



DEFINITE LOOP STRUCTURES

```
final initial   DO   ---   LOOP

final initial   DO   ---   increment +LOOP
```

Functions:

DO	remove 2 parameters, set loop index to initial
LOOP	add 1 to index. Exit loop if index=> final. Otherwise, branch back to DO.
+LOOP	add increment to index. Same exit condition.

OTHER DO-LOOP INSTRUCTIONS

I	current loop index
J	next outer loop index
LEAVE	set current loop index to limit and exit next time at LOOP or +LOOP .

Note:

DO , LOOP , +LOOP , I , J , and LEAVE can be used only within : definitions.

EXAMPLES OF DO LOOPS

```
: COUNT   DO I . LOOP ;   ok
4 0 COUNT   0 1 2 3 ok
0 4 COUNT   4 ok
-16 -20 COUNT   -20 -19 -18 -17 ok

: 2+COUNT   DO I . 2 +LOOP ;   ok
10 0 2+COUNT   0 2 4 6 8 ok
9 1 2+COUNT   1 3 5 7 ok

: 10-COUNT   DO I . -10 +LOOP ;
50 100 10-COUNT   100 90 80 70 60 ok
```

EXAMPLES OF DO LOOPS

```
: INC-COUNT   DO I . DUP +LOOP   DROP ;
1 5 0 INC-COUNT   0 1 2 3 4 ok
2 5 0 INC-COUNT   0 2 4 ok
-3 -10 5 INC-COUNT   5 2 -1 -4 -7 ok

: +COUNT   DO I . I 0=   IF LEAVE THEN   LOOP ;   ok
5 1 +COUNT   1 2 3 4 ok
5 -3 +COUNT   -3 -2 -1 0 ok
```

NON-DESTRUCTIVE STACK PRINT

```
: DEPTH   S0 @   'S -   2/ 2- ;   ok

: .S   DEPTH IF   'S   S0 @ 4 -
                  DO I @ . -2 +LOOP
             ELSE ." Empty"   THEN   ;

1 2 3 .S   1 2 3 ok
.S   Empty ok
```

INDEFINITE LOOPS

--- BEGIN loop_body value UNTIL ---

Functions:

loop_body is always executed once.

UNTIL remove value, exit loop if value is 0 (true), or
branch to BEGIN if value is not 0 (false).

Note: BEGIN and UNTIL can be used only within : definitions

EXAMPLES OF INDEFINITE LOOPS

```
: COUNT-DOWN   BEGIN   DUP . 1-   DUP 0= UNTIL   DROP ;

5 COUNT-DOWN   5 4 3 2 1 0 ok

: HALVES   BEGIN   DUP . 2/   DUP 0= UNTIL   DROP ;

16 HALVES   16 8 4 2 1 0 ok
```

A MORE GENERAL INDEFINITE LOOP

BEGIN loop_body_1 value WHILE loop_body_2 REPEAT

Functions:

loop_body_1 is executed at least once.

WHILE remove value. Exit loop if value is 0 (false), or execute loop_body_2 then branch to BEGIN.

REPEAT branch back to BEGIN .

Note: BEGIN , WHILE and REPEAT can be used only in : definition

SESSION V. UTILITIES IN FORTH

OBJECTIVE
Learn to use FORTH editor to create large programs on disk, and FORTH assembler to create machine level instructions.

TOPICS
- Editor instructions set
- Virtual memory
- Disk input and output
- Program saving and loading
- FORTH assembler
- Memory access and device control

FORTH ASSEMBLER

CODE name assembly_instructions END-CODE

Attributes:

- Full machine speed
- Access to all hardware resources
- Interface exactly like : words with universal reference and stack arguments.
- Macro capability
- Structured programming control structures
- Full support by the resident FORTH system

FORTH ASSEMBLER

CODE name assembly_instructions END-CODE

Functions

CODE create dictionary head for name and invoke the assembler vocabulary.

assembly_instructions
assemble machine codes into the parameter field of the code instruction 'name'.

END-CODE complete the code instruction and make it available for execution or compilation.

CODE INTERPRETER

The code interpreter must be the last executable code in the code definition to execute the next instruction in sequence.

NEXT	execute the next instruction whose address is contained in the interpretive register IP.
POP JMP	discard top of stack and jump to NEXT .
PUSH JMP	push register 0 onto stack and jump to NEXT .
PUT JMP	store register 0 into top of stack and jump to NEXT .
WAIT	jump to multitasker like PAUSE.

ASSEMBLER INSTRUCTIONS

Instruction without operands:

HALT , TRAP , CLC , SEC , ...

Single operand instructions:

CLR , COM , INC , ASR , ASL , ...

Double operand instructions:

MOV , ADD , SUB , BIT , BIC , XOR , ...

FORTH REGISTERS

Machine register	FORTH register	Function
0		scratch
1		scratch
2	W	current word pointer
3	U	user area pointer
4	IP	interpretive pointer
5	SP	data stack pointer
6	RP	return stack pointer
7	PC	program counter

ADDRESSING MODES

Immediate data

```
CODE ONE    S ) 1 # MOV    NEXT    END-CODE
```

Relative addressing

```
CODE NEGATE    S ) NEG    NEXT    END-CODE
```

Post-increment relative

```
CODE DROP    S )+ TST    NEXT    END-CODE
```

Pre-decrement relative

```
CODE DUP    S ) S -) MOV    NEXT    END-CODE
```

Indexed relative

```
CODE SWAP    0 2 S) MOV    2 S) S ) MOV    PUT JMP    END-CODE
```

CONTROL STRUCTURES IN FORTH ASSEMBLER

```
flag    IF    true_part    ELSE    false_part    THEN
```

```
BEGIN    loop_body    flag    UNTIL
```

Flags:

```
0<      negative flag set
0>      negative flag clear
0=      zero flag set
CS      carry flag set
VS      overflow flag set
```

FORTH ASSEMBLER MACROS

Definition:

```
: macro_name    assembler_instructions    ;
```

Examples:

```
: NEXT    W IP )+ MOV    W )+ ) JMP    ;
: BEGIN    HERE    ;
: UNTIL    NOT SWAP HERE 2+    - 2/ 377 AND    + ,    ;
```

VIRTUAL MEMORY

Mass storages are divided into blocks of 1024 bytes. The blocks are numbered consecutively and are accessed by the block numbers

A number of disk buffers (1024 bytes long) are reserved in the RAM memory for temporarily storing data in disk blocks.

Disk data are accessed in the disk buffers using memory access instructions.

Modified blocks in disk buffers are written back to disk when buffers are reassigned to other blocks or when FLUSH is invoked.

ACCESSING THE VIRTUAL MEMORY

BLOCK (n --- addr) is the fundamental tool to access the virtual memory. Its functions are:

- Check the disk buffers to see if Block n is in one of the buffers. If so, return the buffer address.
- Block n is not in the buffers. Choose the lest accessed buffer for Block n. Flush the contents of this buffer back to the mass storgae if it was marked as updated.
- Read Block n from mass storage to this buffer, and put its address on the stack.

With the buffer address, the user will be able to read or write storage using memory accessing instructions.

ACCESSING THE VIRTUAL MEMORY

BUFFER (n --- addr)

- Select the lest accessed disk buffer for Block n. If the contents of this buffer was marked as updated, flush it back to the mass storage.
- Return the buffer address on the stack.

The difference between BUFFER and BLOCK is that BUFFER will not read data from the mass storage. It is used to store raw data into the mass storage. BLOCK is used to update and used the data already stored in the mass storage.

DATA INTEGRITY

- Data can be read or written freely in the disk buffers. Any changes may in the buffer are not written back to the mass storage immediately.
- If it is desired that modified data are stored into the mass storage, the instruction UPDATE must be given after the modification.
- Modified data in the updated buffer are written back to the mass storage only when:
  - The buffer will be used by another block as commanded by BLOCK or BUFFER.
  - Explicit write instruction FLUSH is executed.

DATA INTEGRITY

- Buffers can be scratched by EMPTY-BUFFERS, if data in the buffers are known to be disturbed. EMPTY-BUFFERS clears all the disk buffers so that the data on the mass storage are not changed.
- The set of instructions BLOCK, BUFFER, UPDATE, and EMPTY-BUFFERS provides a very efficient means to use large volume mass storage with a high degree of data security.

SESSION VI. FORTH VIRTUAL COMPUTER

OBJECTIVE
Learn the internal structures of a virtual FORTH computer, and the detailed mechanism of its operations.

TOPICS
- Registers and pointers
- Data and return stacks
- Dictionary
- Disk and terminal buffers
- Inner interpreters
- Machine instructions interpreter
- High level instruction interpreter
- Constants and variables

REGISTERS IN THE FORTH VIRTUAL COMPUTER

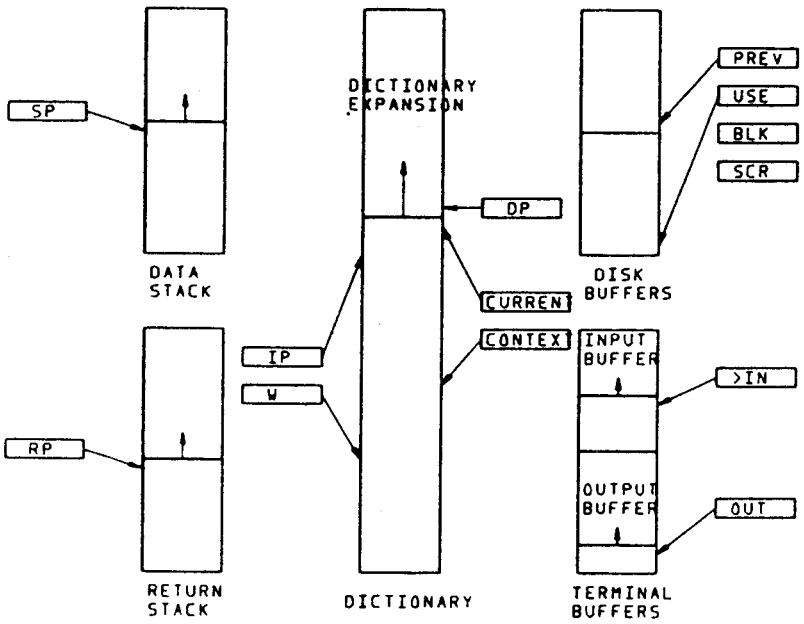
. Data stack pointer	SP
. Return stack pointer	RP
. Instruction pointer	IP
. Current word pointer	W
. User area pointer	UP

INNER INTERPRETERS

The inner interpreters are the machine code routines which executes different types of instructions in the FORTH computer.

The interpreter for the low level FORTH instructions:

```
NEXT:    MOV   (IP)+,W      Pop the address of the next
                            instruction in to W register.
                            Increment IP, pointing to the
                            next instruction to be executed.
         JMP   @(W)+        Jump indirectly through the
                            address pointed to by W.
                            Increment W, pointing to the PFA
                            of the word being executed.
```



12. The Virtual FORTH Computer

INNER INTERPRETERS

NEXT at the end of every code instruction pulls in the next instruction to be executed. To start the chain of execution, the instruction EXECUTE is used. EXECUTE assumes that the execution address of the instruction to be executed is on the data stack.

```
EXECUTE:    MOV    (SP)+,W      Pop the execution address from the
                                data stack to W register.
            JMP    @(W)+        Execute the instruction by jumping
                                indirectly through its execution
                                address, which points to the
                                machine code routine to be
                                executed.
```

ADDRESS INTERPRETER

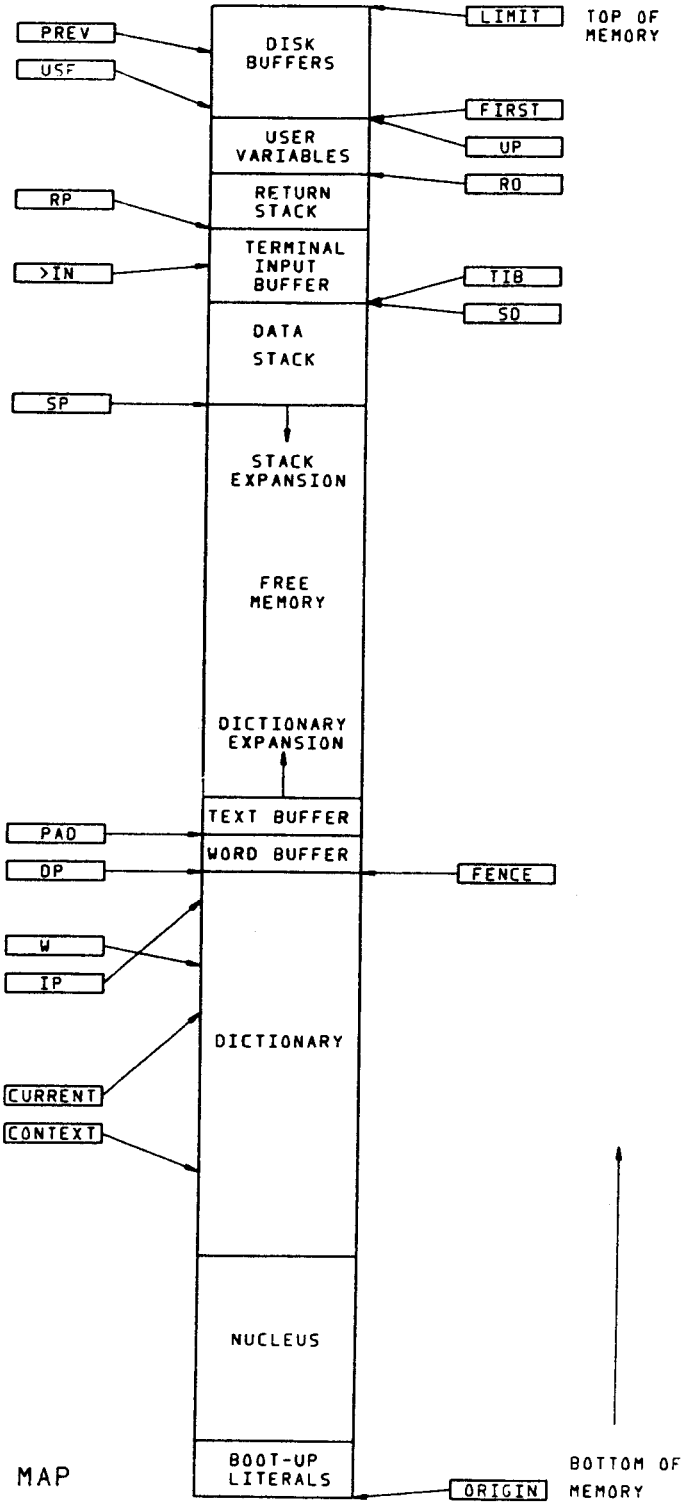
High level FORTH instruction contains a list of execution addresses to be executed in sequence. The address interpreter processes this list of addresses and execute them in sequence.

```
DOCOL:      MOV    IP,-(RP)     Push the next execution address
                                on the return stack.
            MOV    W,IP         Point IP to the list of execu-
                                tion addresses of this high
                                level instruction.
            MOV    (IP)+,W      Execute it by calling NEXT.
            JMP    @(W)+
```

ADDRESS INTERPRETER

At the end of the address list in every high level FORTH instruction, there is an instruction which returns the control to the calling instruction, very similar to the RETURN in FORTRAN.

```
EXIT:       MOV    (RP)+,IP     Pop the next execution address
                                from the return stack back into
                                the instruction pointer IP.
            MOV    (IP)+,W      Call NEXT to execute it.
            JMP    @(W)+
```



THE MEMORY MAP

13. Memory Map and Pointers

CONSTANT INTERPRETER

The execution address in a constant instruction is pointing to a code routine which pushes the integer number stored in the constant instruction on to the data stack, achieving the function of a constant. This code routine is the interpreter for all constants used in FORTH system.

```
DOCON:      MOV    (W),-(SP)      Push the constant onto the data
                                  stack.
            MOV    (IP)+,W        Call NEXT to continue the
                                  execution sequence.
            JMP    @(W)+
```

VARIABLE INTERPRETER

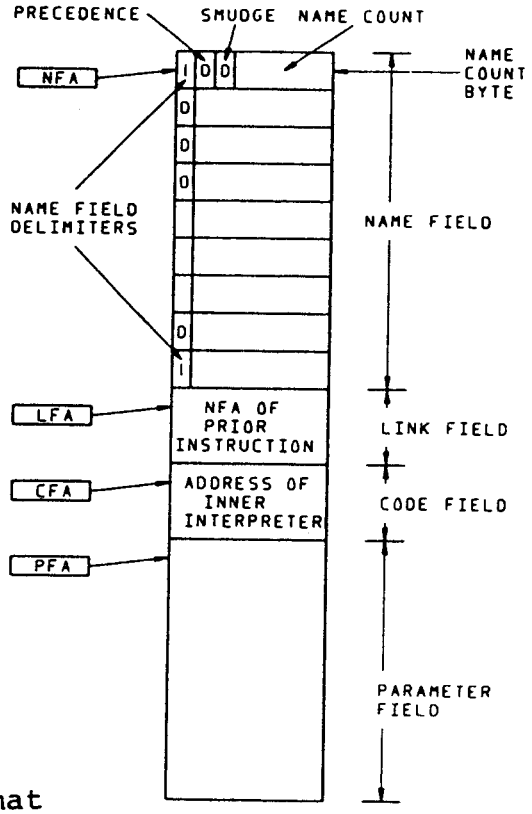
The code routine which interprets all the variables defined in FORTH pushes the address of the cell where the number is stored onto the data stack.

```
DOVAR:      MOV    W,-(SP)        Push the address of the variable
                                  onto the data stack.
            MOV    (IP)+,W        Call NEXT.
            JMP    @(W)+
```

USER VARIABLE INTERPRETER

The operational environment of each user in a multitasking FORTH system are defined by a set of user variables private to each user. These variables are accessed through the user area pointer, allowing users to share a common dictionary of instructions.

```
DOUSE:      MOV    (W),-(SP)      Push the user area offset onto
                                  the data stack.
            ADD    UP,(SP)        Add the user area pointer to
                                  get the address of the variable.
            MOV    (IP)+,W        Call NEXT.
            JMP    @(W)+
```



14. Instruction Format

POINTERS IN THE FORTH SYSTEM

Many pointers are needed by the FORTH system to manage the resources and to perform system functions.

DP	Top of dictionary and the word buffer.
PAD	Text input/output buffer.
TIB	Terminal input buffer.
PREV, USE	Disk buffer pointers.
CONTEXT	Pointer to the end of the context vocabulary.
CURRENT	Pointer to the end of the current vocabulary.
BLK	The block number of the disk block under processing.
>IN	The character pointer for text parsing.

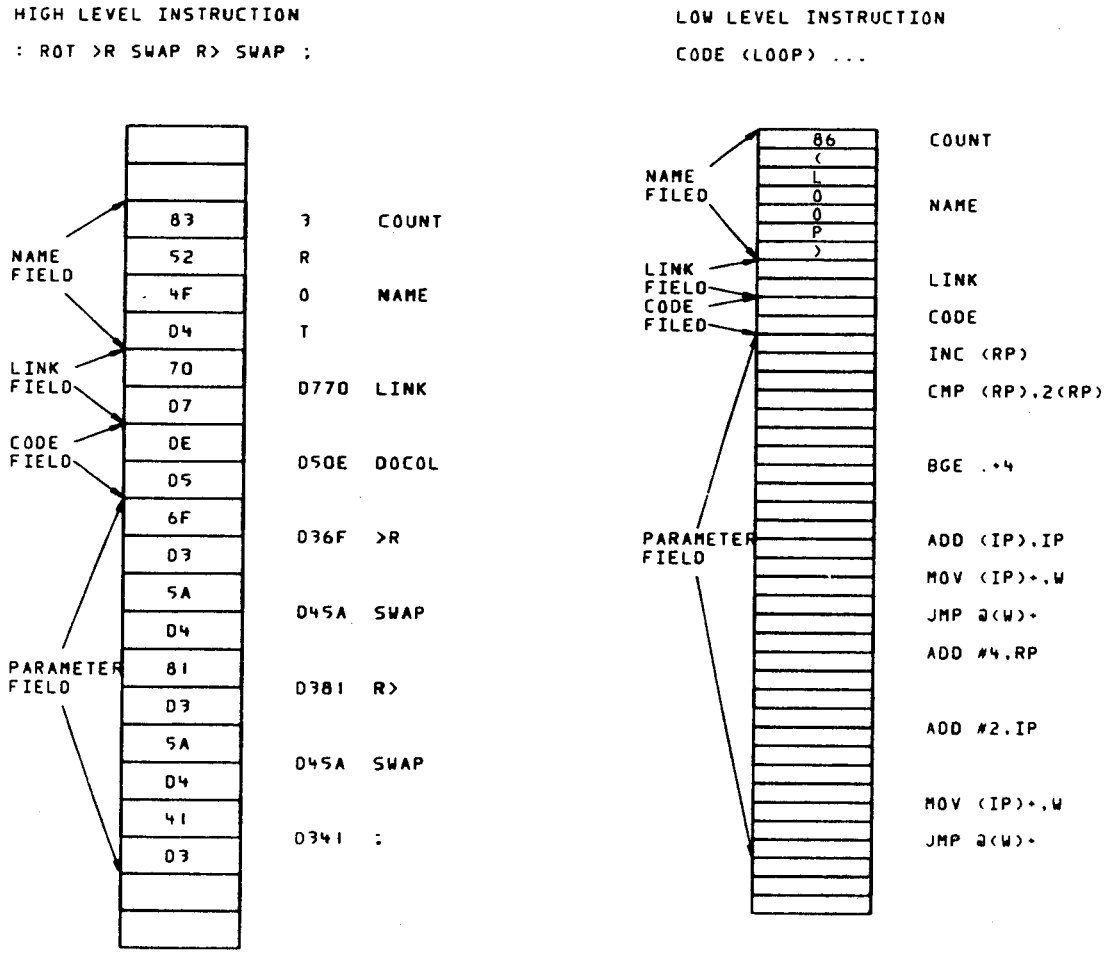
INSTRUCTION FORMAT

All FORTH instructions share a regular structure of format. There are four fields in an instruction:

Name Field	Variable length. The first byte is the length byte. The ascii name can have up to 31 character
Link Field	Pointing to the name field of the prior instruction in the same vocabulary.
Code Field	Pointing to the code routine, the interpreter of this instruction.
Parameter Field	Variable length. Containing necessary data to perform the function designed of this instruction.

DICTIONARY ORGANIZATION

- The dictionary is basically a linearly linked list of instructions.
- Branches or vocabularies are allowed in the dictionary, so that related instructions can be linked for specific searching mechanism. Instructions must be uniquely identifiable by their names and the vocabulary they belong.
- CONTEXT and CURRENT are used to specify vocabularies to be searched by the text interpreter.



15. High Level and Low Level Instructions

SESSION VII. FORTH OPERATING SYSTEM

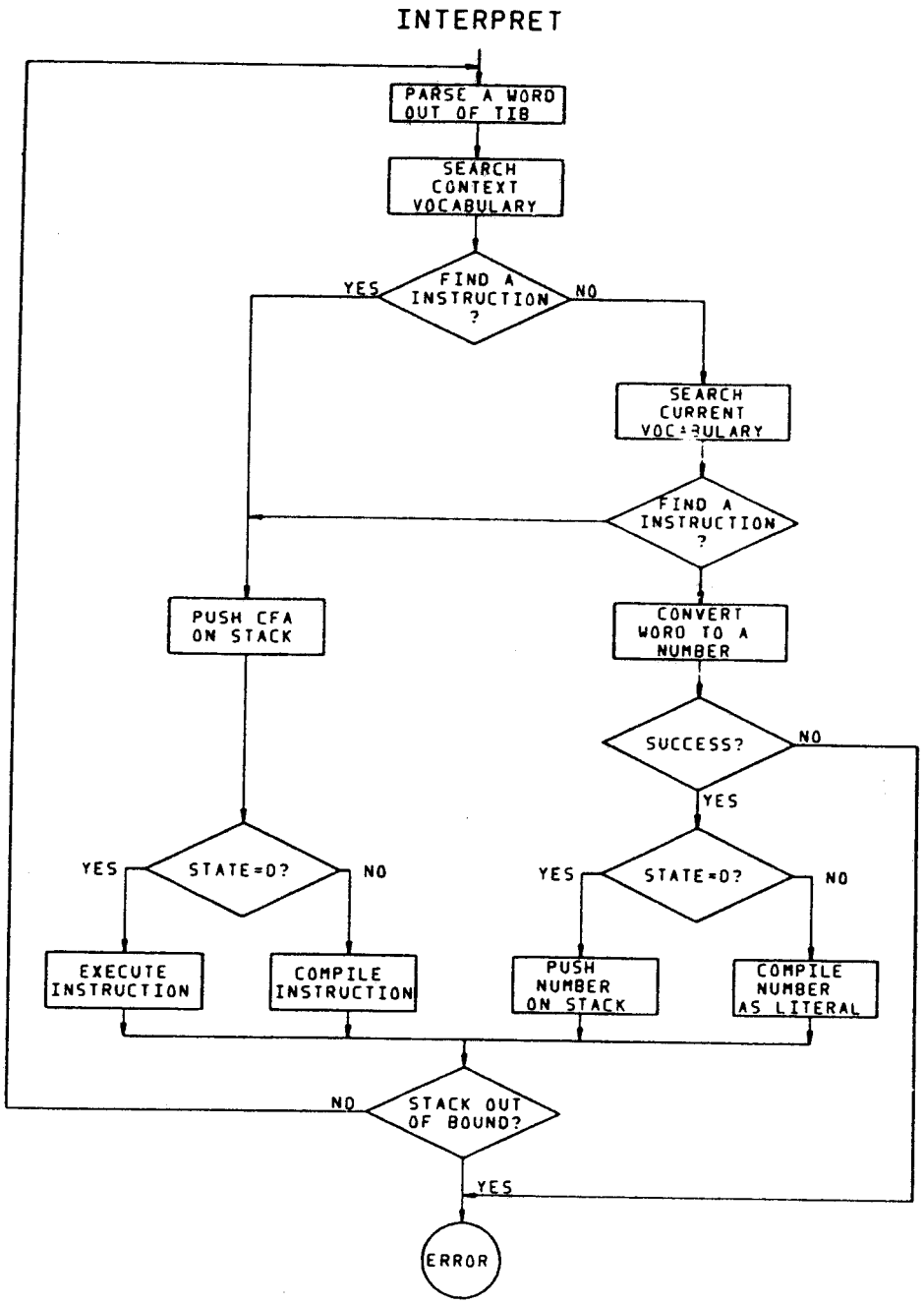
OBJECTIVE
Learn the detailed functions of the text interpreter and its interaction with user.

TOPICS
- FORTH main loop
- Text interpreter loop
- Interpreter and compiler
- Dictionary and vocabulary
- Number conversions
- Error handling

```
( COLD START )
: COLD    ( COLD START, INITIATE USER AREA )
       EMPTY-BUFFERS    FIRST USE !    FIRST PREV !
       18 +ORIGIN    S0 24 CMOVE    ( MOVE 24 BYTES )
       14 +ORIGIN @    FORTH 6 +    !    ( STORE VOC-LINK )
       ABORT    ;

;S
```

```
( QUIT, ABORT )

: QUIT    ( RESTART, INTERPRET FROM TERMINAL )
       0 BLK !    [COMPILE] [
       BEGIN    RP! CR    QUERY INTERPRET
           STATE @    0= IF ."  OK" ENDIF
       AGAIN ;

: ABORT    ( WARM RESTART, INCLUDING REGISTERS )
       SP!    DECIMAL    DR0
       CR    ." FIG-FORTH"
       [COMPILE] FORTH DEFINITIONS    QUIT    ;

;S
```

```
( INTERPRET )

: INTERPRET    ( INTERPRET OR COMPILE SOURCE TEXT INPUT WORDS )
      BEGIN    -FIND
          IF ( FOUND )    STATE @ <
              IF CFA ,    ELSE CFA EXECUTE    ENDIF    ?STACK
          ELSE    HERE NUMBER    DPL @ 1+
              IF [COMPILE] DLITERAL
              ELSE DROP    [COMPILE] LITERAL    ENDIF    ?STACK
          ENDIF
      AGAIN ;

;S




( LITERAL, DLITERAL, [COMPILE], U< , ?STACK )

: [COMPILE]    ( COMPILE FOLLOWING IMMEDIATE WORD )
      -FIND    0= 0 ?ERROR    DROP CFA ,    ;    IMMEDIATE

: LITERAL    ( IF COMPILING, CREATE LITERAL )
      STATE @    IF COMPILE LIT ,    ENDIF    ;    IMMEDIATE

: DLITERAL    ( IF COMPILING, CREATE DOUBLE LITERAL )
      STATE @    IF SWAP    [COMPILE] LITERAL
                    [COMPILE] LITERAL    ENDIF    ;    IMMEDIATE

: U<    >R 0 R> 0    DMINUS D+    SWAP DROP    0<    ;

: ?STACK    ( CHECK STACK OVERFLOW OR UNDERFLOW )
      S0 @ 2 - SP@ U< 1 ?ERROR SP@ HERE 128 + U< 2 ?ERROR ;    ;S



( CONVERT, NUMBER, -FIND )
: CONVERT    ( D1 ADDR1 --- D2 ADDR2 )
      BEGIN    1+ DUP >R C@    BASE @    DIGIT
          WHILE    SWAP BASE @ U*    DROP ROT    BASE @    U*    D+
          DPL @ 1+    IF 1 DPL +!    ENDIF    R> REPEAT    R> ;

: NUMBER    ( ADDR --- D )
      0 0 ROT    DUP 1+ C@    45 = DUP    >R + -1
      BEGIN    DPL !    CONVERT    DUP C@ BL -
          WHILE    DUP C@    46 -    0 ?ERROR    0 REPEAT
          DROP R>    IF DMINUS ENDIF ;

: -FIND    ( --- PFA COUNT TF  OR FF )
      BL WORD    HERE CONTEXT @ @    (FIND)
      DUP 0=    IF    DROP HERE LATEST    (FIND)    ENDIF ;
: FIND    -FIND IF DROP ELSE 0 THEN ;    ;S
```



16. Flow Chart of Text Interpreter

```
( ERROR HANDLER )

: (ABORT)    ABORT ;    ( USER ALTERABLE ERROR ABORT )

: ERROR    ( WARNING: -1 ABORT, 0 NO DISC, 1 DISC )
      WARNING @ 0<    ( PRINT TEXT LINE REL TO SCR #4 )
      IF (ABORT) ENDIF    HERE COUNT TYPE    ." ? "
      MESSAGE    SP!    IN @ BLK @    QUIT    ;

: ID.    ( PRINT NAME FIELD FROM ITS HEADER ADDRESS )
      PAD    32 95 FILL    DUP    PFA LFA OVER -
      PAD SWAP CMOVE    PAD COUNT 31 AND    TYPE SPACE ;

;S




( WORD )
: WORD    ( CHAR --- , MOVE STRING TO HERE )
      BLK @    IF BLK @    BLOCK    ELSE TIB @    ENDIF
      IN @ + SWAP    ( ADDR CHAR )
      ENCLOSE    ( ADDR START END COUNT )
      HERE 34 BLANKS    ( CLEAR WORD BUFFER )
      IN +!    ( STEP OVER THIS STRING )
      OVER - >R    ( SAVE CHARACTER COUNT )
      R HERE C!    ( STORE LENGTH BYTE FIRST )
      + HERE 1+
      R> CMOVE ;    ( MOVE STRING FROM BUFFER TO HERE+1 )

;S




( CREATE )

HEX
: CREATE    ( MAKE A SMUDGED CODE HEADER TO PARAMETER FIELD )
      -FIND    ( CHECK IF UNIQUE IN CURRENT AND CONTEXT )
      IF    ( WARN USER )    DROP    NFA ID.
          4 MESSAGE SPACE    ENDIF
      HERE DUP C@    WIDTH @    MIN 1+ ALLOT    ?ALIGN
      DUP A0 TOGGLE    HERE 1 - 80 TOGGLE    ( DELIMIT BITS )
      LATEST ,    CURRENT @ !
      HERE 2+ , ;

DECIMAL ;S
```

SESSION VIII. FORTH PROGRAMMING LANGUAGE

OBJECTIVE
Learn the syntax rules of FORTH with its instruction set, extensions, and language standards.

TOPICS
. Character set
. Words, numbers, and instructions
. Standard instructions
. String instructions
. User instructions
. Defining instructions
. FORTH standards

PROGRAMMING LANGUAGE

A programming language is a set of symbols with rules for combining them to specify execution procedures to a computer or to communicate the problem solving procedures between programmers.

Essence of a programming language:

- Symbols or character set
- Rules or syntax

CHARACTER SET

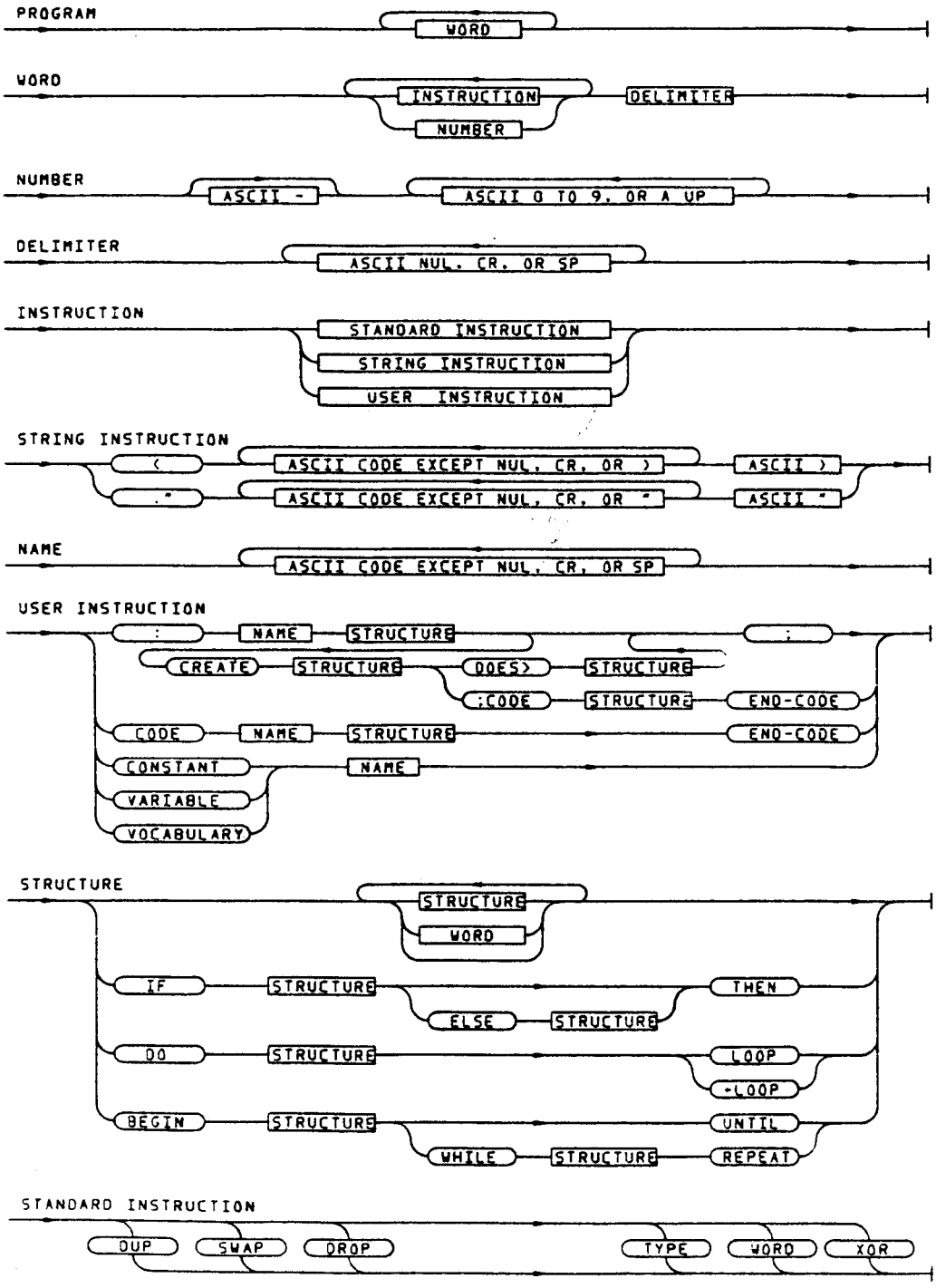
FORTH employs the full ASCII character set:

- Upper and lower case alphabets
- Numerals 0 to 9
- All the printable punctuations and symbols
- All the non-printable control characters

Exceptions:

- Ascii NUL and CR as absolute delimiters
- Ascii SP as default delimiter
- Ascii RUB and BS as backspaces
- Other characters temporarily designated as delimiter

17. Syntax of FORTH



PROGRAM AND WORDS

Program: A list of words, separated by delimiters, to be processed by a FORTH computer.

Word:

- Syntax: A group of characters separated from each other by delimiters.
- Semantics: An instruction or a number.

NUMBER

A group of Ascii characters enumerated from 0 to 9, and then from A up. The number of this set is determined by the current base value used in the system.

Semantics: Numbers are used to represent:

Flags, Ascii characters, bytes, signed and unsigned integers, signed and unsigned double integers, and addresses.

INSTRUCTIONS

Instruction: A named, memory resident, and executable entity in the FORTH computer. It is invoked by its name and the vocabulary it was linked into.

Types of FORTH instructions:

- Standard instructions --- Provided by system.
- User instructions --- Created by the user.
- String instruction --- With special syntax.

STRING INSTRUCTIONS

String instruction processes the character string immediately following it, delimited by a delimiter specified by the string instruction itself.

Two string instructions are generally provided by the system:

(	Comment
."	Print message

STANDARD INSTRUCTIONS

A set of instructions provided by the FORTH system, creating an enviroment in which most programming problems can be conveniently solved.

Standard instructions are usually grouped as:

- Nucleus instructions
- Interpreter instructions
- Compiler instructions
- Extension instructions

USER INSTRUCTIONS

Instructions that can be defined by the user to extend the FORTH system towards the solution to his problem.

User instructions are usually grouped by the defining instructions used to create them:

- High level instructions or Colon instructions
- Low level instructions or code instructions
- Constants
- Variables
- Vocabularies

COLON OR HIGH LEVEL INSTRUCTIONS

A colon instruction is a named equivalence to a list of words. When this instruction is executed, this list of words are actually executed.

A more precise definition of colon instruction:

A colon instruction is a named equivalence to a list of 'structures'.

STRUCTURES

A structure is a list of words and/or structures, which has generally only one entry point and one exit point during execution.

A structure may include:

- Nothing
- A word/structure list
- IF-THEN-structure
- DO-LOOP-structure
- BEGIN-UNTIL-structure

DEFINING WORDS---COMPILER/INTERPRETERS

FORTH provides an unique facility to create new classes of instructions or data structures with user designated execution procedures.

Three steps in using this facility:

1. Create a defining word --- a compiler-interpreter pair.
2. Use the compiler to define new words.
3. Use the interpreter to execute the new word.

: AS A DEFINING WORD

: (Colon) defines high level FORTH instructions.

Example:

```
: ?   @ . ;
```

It compiles a new word ? into the dictionary when : is executed. When the new word ? is executed, the functions of @ and . are executed in sequence.

VARIABLE AS A DEFINING WORD

VARIABLE itself is created by : as follows

```
: VARIABLE   CREATE 2 ALLOT   DOES>   ;
```

Functions:
Compiler---create a new name in dictionary and allocate 2 bytes
for the parameter field to store data.
Interpreter---leave the parameter field address on the stack so
that the data can be read or modified.

```
VARIABLE X   ok
3 X !   ok
X ?   3 ok
```

DEFINING 1 DIMENSIONAL BYTE ARRAY

Create the array defining word:

```
: CVECTOR   CREATE ALLOT   DOES>   +   ;   ok
```

Create a 5 byte array:

```
5 CVECTOR LINE   ok
```

Usage:

```
: 0LINE   5 0 DO   0 I LINE C!   LOOP   ;
: .LINE   5 0 DO   I LINE C@ .   LOOP   ;
: ILINE   5 0 DO   I I LINE C!   LOOP   ;
```

VARIATIONS ON THE BYTE ARRAY

Symmetric subscripts:

```
: CVECTOR   CREATE ALLOT   DOES>   + 1-   ;   ok
```

Clear the array as defined:

```
: CLEAR   DUP HERE SWAP ERASE   ALLOT ;
: CVECTOR   CREATE CLEAR   DOES>   +   ;
```

VARIATIONS ON THE BYTE ARRAY

Add range check:

```
: CVECTOR    CREATE    DUP , ALLOT
             DOES>     2DUP    @ < IF +    ELSE ." RANGE ERROR"
                       . . THEN    ;
```

Pure code in ROM:

```
: CVECTOR    CREATE    THERE , ALLOT    DOES>    @ +    ;
```

DEFINING VIRTUAL ARRAY ON DISK

```
: CVECTOR    CREATE    DISK-DP@ , DISK-ALLOT
             DOES>     DUP ROT 1024 /MOD
                       ROT +    BLOCK +    ;
```

UPCOUNTER

```
: UPCOUNTER    CREATE ,
               DOES>    1 OVER    +!    ;

0 UPCOUNTER LINE#

LINE# ?    1 ok
LINE# ?    2 ok
LINE# ?    3 ok
```

```
( MESSAGE OUTPUT)

Compile a named message.  Type the message by invoking the name.

: MESSAGE    CREATE     ( build the head )
             1 WORD     ( Accept the following string to body.)
             HERE C@    ( The length byte of the string.)
             1+ ALLOT   ( The parameter field.)
             DOES>
             COUNT TYPE ;  ( Type out the string.)


MESSAGE HELLO   HOW ARE YOU?   ok
MESSAGE ANSWER   I AM FINE.  AND YOU?   ok
HELLO   HOW ARE YOU? ok
ANSWER   I AM FINE.  AND YOU? ok



( ASCII MESSAGES)

( This is the poly-FORTH message compiler/interpreter.)
( Any ASCII character can be compiled, including NUL.)

: MSG   CREATE    ( Create the head only.  Parameters will be )
                  ( explicity compiled by the user.)
        DOES>     COUNT TYPE ;

HEX
MSG PAGE 0C01 ,    ( Compile byte count 1 and ASCII code 12.)
MSG NAME 5404 , 4E49 , 2047 ,
DECIMAL

PAGE
NAME   TING ok



( DOUBLE NUMBER CONSTANTS AND VARIABLES)


: 2CONSTANT   CREATE    , , ( Compile two integers off stack.)
              DOES>     2@  ;     ( Push it on stack.)

: 2VARIABLE   CREATE  4 ALLOT ( Allot extra 2 bytes.)
              DOES>   ;   ( The address is on stack already.)


1.23456 2CONSTANT RATIO   ok
RATIO D.   123456 ok
2VARIABLE PAY   ok
RATIO PAY 2!   ok
PAY 2@ D.   123456 ok
```

SESSION IX. FORTH PROGRAMMING STYLE

OBJECTIVE
Learn the stylistic problems in writing and documenting large FORTH programs and projects.

TOPICS
- Source code layouts
- Modular structure in blocks
- Reducing program complexities
- Factorization of modules
- Layered program structures
- Top-down program design
- Program documentation standards

STYLISTIC CONSIDERATIONS IN FORTH PROGRAMMING

Some materials in this session are adopted from Kim Harris' paper on 'Software quality & FORTH Programs', 1980 FORML Proc., p. 140-172.

The measurment on software quality is based upon following considerations:

PERFORMACE	Execution time, memory size, etc.
MAINTAINABILITY	Documentation, modifications, etc.

PERFORMANCE

EXECUTION TIME can be reduced by:
- Rewriting critical routines in assembly codes.
- Bundling high level words.

MEMORY SIZE can be reduced by
- Factorizing reusable phrases into new words.
- Dynamic memory allocation using stack or PAD.
- Invoke virtual memory for data storage.

Execution time and memory size are the two conflicting factors which have to be traded off one against the other.

SOFTWARE MAINTAINANCE --- DOCUMENTATION

EXTERNAL (USER) DOCUMENTATIONS

- Specification and overview
- User manuals
- Help screens and help commands
- Input prompts

INTERNAL (SYSTEM) DOCUMENTATIONS

- Implementation overview
- Glossary in alphabetic and functional order.
- Source listings

SOURCE LISTING ORGANIZATIONS

Group related items together on listings

- Screen = paragraph
- Triad = page
- 30 Screens = chapter

Configuration control

- Keep date and author of last change on each screen
- Use load screens as contents of applications

LAYOUT OF SOURCE ON A SCREEN

- Include only related items on one screen
- Use line 0 as a comment with contents, author and date
- Leave line 15 blank for future expansion
- Only one definition per line
- Specify stack effects after the name of definition
- Separate phrases with 3 spaces

PROGRAMMING FOR MAINTAINABILITY

Avoid side effects by:

- Use structured programming restrictions
- Avoid global references to variables and data
- Factoring incidental or unrelated functions into separated definitions

Side effects are hard to document, hard to understand, and not obvious. These are the bugs which refuse to be flushed out. They are to be avoided as plagues.

FACTORING VERSUS BUNDLING

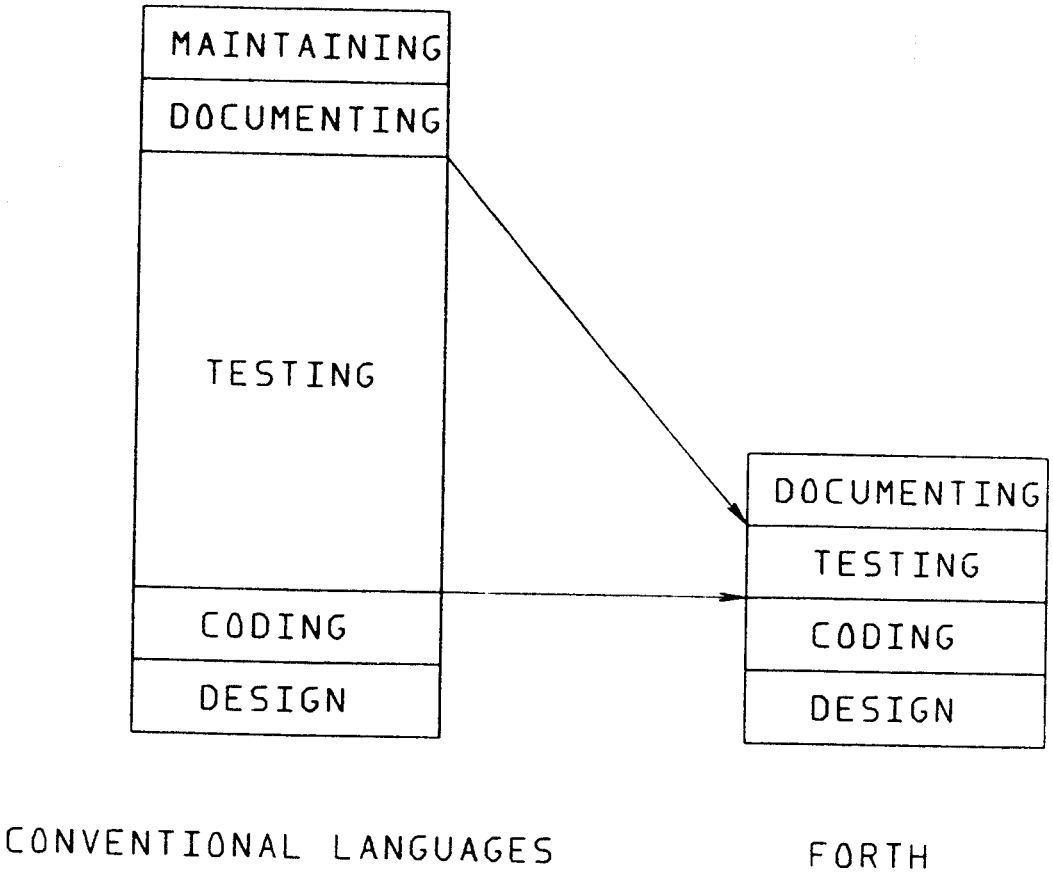
The trade off between factoring and bundling:

- One function per defintion enhances maintainability.
- Reusable defintions reduce memory size.
- Bundling functions together reduces execution time.
- Bundling functions reduces the number of names.

FACTOR LONG DEFINITIONS

In cases where the following considerations are important, factor long defintions into short definitions:

- Reduce internal complexity
- Reusability
- Transportability



18. **Software Costs**

PROGRAMMING COST REDUCTION

The cost of programming is directely related to its complexity. Minimizing program complexity can be translated to cost reduction.

TECHNIQUES FOR MINIMIZING COMPLEXITY

- Decompose problems into sub-problems or modules
- Minimize interface complexity between modules by restricting data flow between them.
- Minimizing complexity within modules using control structures.

With the simplest syntax and internal structures, FORTH is the best software tool to arrive at the simplest solutions.

THE FORTH COMMANDMENTS

I. WORDS MUST BE SEPARATED BY SPACES OR CARRIAGE RETURNS.

II. A WORD MUST BE UNIQUE IN ITS FIRST 3 (OR 31) CHARACTERS, THE CHARACTER COUNT, AND THE VOCABULARY IT IS LINKED.

THE FORTH COMMANDMENTS

III. ALL INSTRUCTIONS MUST REMOVE THEIR PARAMETERS FROM THE DATA STACK, AND LEAVE ONLY EXPLICIT RESULTS ON THE STACK.

IV. THE RETURN STACK MUST BE RESTORED AT EVERY LEVEL OF DO-LOOP AND BEFORE THE END OF AN INSTRUCTION.

THE FORTH COMMANDMENTS

V. STRUCTURE INSTRUCTIONS MUST BE PROPERLY PAIRED.

VI. STRUCTURES CAN BE NESTED, BUT NOT OVERLAPED.

THE FORTH COMMANDMENTS

VII. NUMERIC DATA MUST PRECEDE THE INSTRUCTION WHICH USES THEM.

VIII. STRING INSTRUCTION MUST BE FOLLOWED BY THE STRING TO BE PROCESSED.

THE FORTH COMMANDMENTS

IX. COMMENT ON THE FIRST LINE IN A SCREEN.

X. COMMENT ON THE STACK EFFECTS OF A DEFINITION.

THE FORTH COMMANDMENTS

XI. PUT ONLY RELATED WORDS IN ONE SCREEN.

XII. DO NOT OVERPACK SCREENS.

THE FORTH COMMANDMENTS

XIII. DEFINE SHORT DEFINITIONS.

XIV. FACTOR OUT REUSABLE FUNCTIONS.