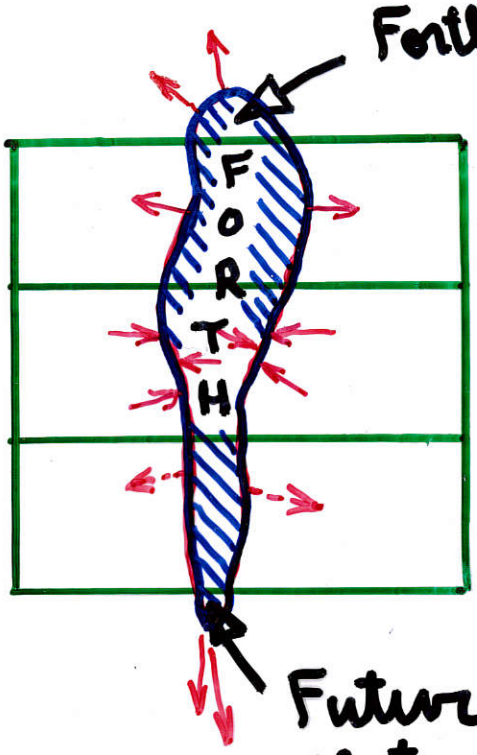
Languages:

low-level

middle-level
Pascal, C, Fortran

very high-level
Prolog, LISP, Smalltalk

as in JFAR v4#4
(R.D.Dixon)



$TX-2000
$TX-4000

Imperative style
Functional style
Object-oriented style
Programming in logic

\+
\+

extensible to support
as implementation tool

System programming
Real-time applications
Expert systems, AI
Translating
Databases

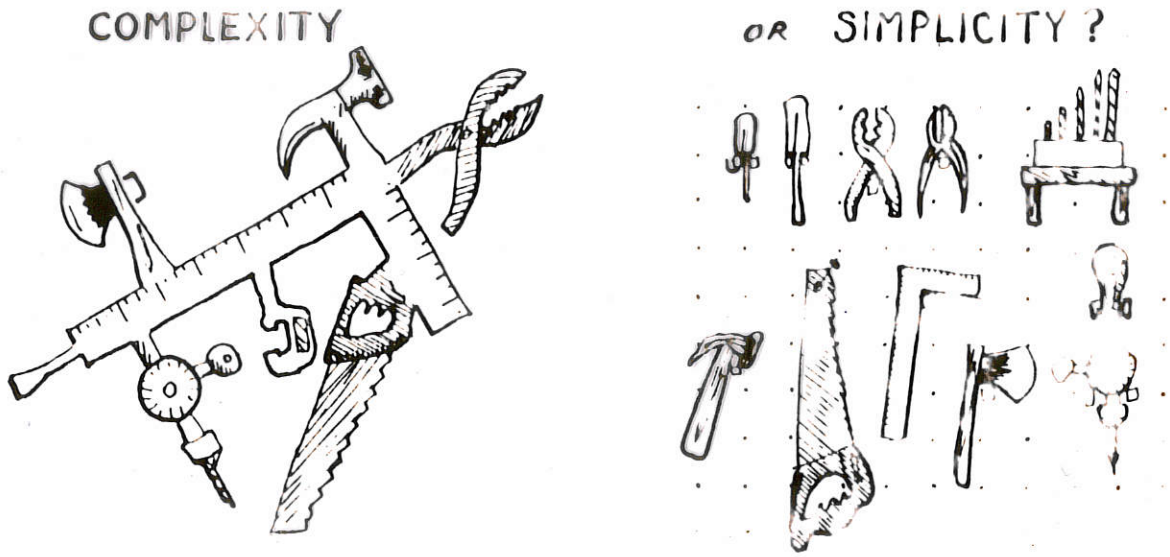
Compare this to the diagram of INTERPRET and you'll see that] could be called an interpreter with the ability to decide whether to execute or to compile any given word. It is the simplicity of this design that lets you add new compiling words so easily.

In summary, we've shown two ways to extend the FORTH compiler:

1. Add new, specialized compilers, by creating new defining words.

2. Extend the existing colon compiler by creating new compiling words.

While traditional compilers try to be universal tools, the FORTH compiler is a collection of separate, simple tools ... with room for more. Which approach seems more useful:



Leo Brodie

- Starting Forth
- Thinking Forth
- Mastering Forth

Loeliger R.G. Threaded interpretive languages
McCabe C.K. Forth Fundamentals I II
Forth-83 Standard

Forth → Language multi-level
Forth → Integrated system

Forth-system:
* Kernel (~4-6 K)
  - code primitives (~1-3 K)
  - secondaries in macro Assembler
* Assembler (special Forth-assembler)
  (4-16 K machine dependent)
* Editor (1-10 K)
* Packages
  - graphics
  - floating point
  - mouse, plotter e.t.c.
  - processes, multi-using
  - data base management

Dictionary of words

Text interpreter (outer interpreter)
Threaded list (threaded code)
Code interpreter (inner interpreter)
Return-stack
Parameter stack (computation stack)
Block buffers

Program size: ≈ 1.5 × more compact than machine code!

Speed: 3-4 × slower than machine code.

FORTH Ch. Moore USA ~1969

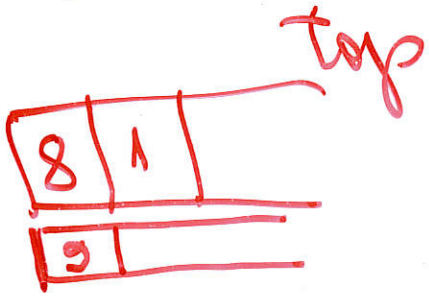
* Extensible, opened. It's possible to build up a problem-oriented virtual FORTH-machine.
* Widely spread. FORTH-83 standard.
* Compact, laconic. allows to work with poor and special hardware.
* Flexible. It's rather easy to reach a good mobility of software.
* Easy to implementate. Real FORTH-oriented hardware.
* Stack-oriented, highly modular.

CONTRA:

* Interpretive (mainly).
* Hard to read (write-only), postfix notation is unusual.
* all the kernel is needed to run a certain program.

Postfix notation (reverse Polish).

1) $8 + 1$ infix
$8\ 1\ +$

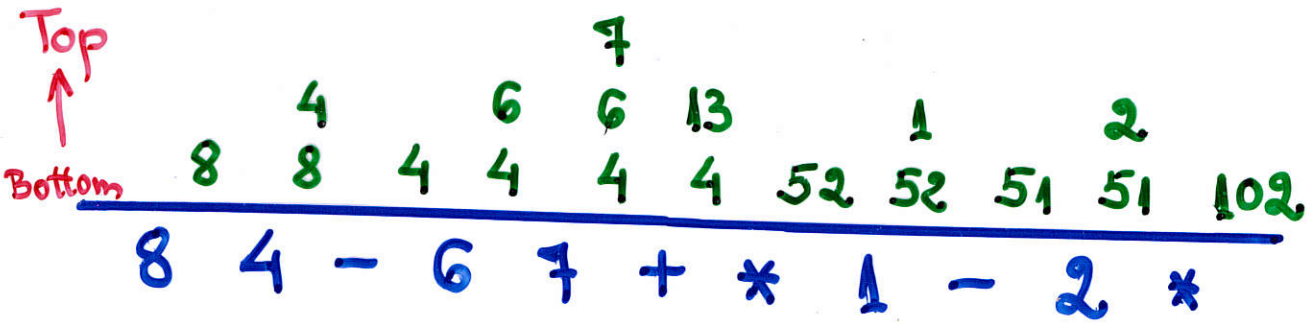


2) $6 - 4$
$6\ 4\ -$ not commutative!

3) $((8-4)*(6+7)-1)*2$
$8\ 4\ -\ 6\ 7\ +\ *\ 1\ -\ 2\ *$
no brackets!

Number - operation which puts the number value onto the parameter stack.

Binary operations - take 2 topmost elements, perform an operation, leave the result on the stack.



Infix : print 20/4

The order of numbers stays the same. Let's try a division problem:

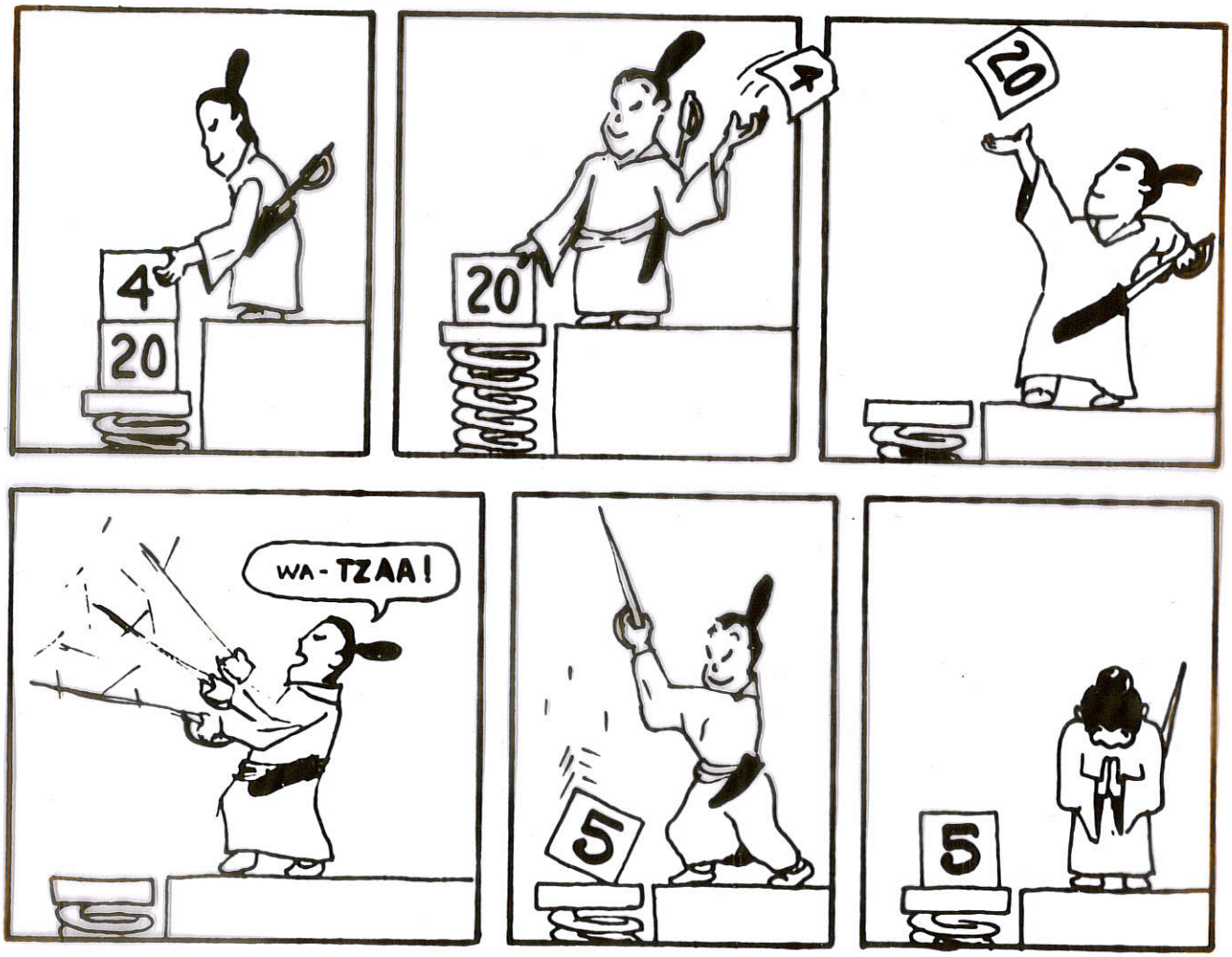
```
20 4 / . 5 ok
```

." Text "

The word / is defined to divide the second number on the stack by the top number:

SAMURAI DIVIDER

(commentary)



What do you do if you have more than one operator in an expression, like:

```
4 + (17 * 12)
```

you ask? Let's take it step-by-step: the parentheses tell you to first multiply seventeen by twelve, then add four. So in FORTH you would write:

```
17 12 * 4 + . 208 ok
```

4 17 12 * + .

and here's why:

Colon definitions.

: ␣ name ␣ contents ␣ ;

```
: SUM3 ( a b c --- sum )
  + + ;

5 8 2 SUM3
  5  5  5  5  15
     8  8  10
        2
```

Bottom ↓ Top

```
CUBE ( a --- a³ )
```

Problem: We have only ONE copy of a.

Solution: stack operations.

```
DUP ( n --- n n )

: CUBE ( a --- a³ )
  DUP DUP * * ;

4 CUBE
  4  4  4  4  64
     4  4  16
        4
```

↓ Top

Compilation and execution - two main states of the text interpreter.

Here's a list of the FORTH words we've covered in this chapter:

+	(n1 n2 -- sum)	Adds.
-	(n1 n2 -- diff)	Subtracts (n1-n2).
*	(n1 n2 -- prod)	Multiplies.
/	(n1 n2 -- quot)	Divides (n1/n2).
/MOD	(u1 u2 -- u-rem u-quot)	Divides. Returns the remainder and quotient.
MOD	(u1 u2 -- u-rem)	Returns the remainder from division.
SWAP	(n1 n2 -- n2 n1)	Reverses the top two stack items.
DUP	(n -- n n)	Duplicates the top stack item.
OVER	(n1 n2 -- n1 n2 n1)	Makes a copy of the second item and pushes it on top.
ROT	(n1 n2 n3 -- n2 n3 n1)	Rotates the third item to the top.
DROP	(n --)	Discards the top stack item.
2SWAP	(d1 d2 -- d2 d1)	Reverses the top two pairs of numbers.
2DUP	(d -- d d)	Duplicates the top pairs of numbers.
2OVER	(d1 d2 -- d1 d2 d1)	Makes a copy of the second pair of numbers and pushes it on top.
2DROP	(d --)	Discards the top pair of numbers.

Simple stack-arithmetics (integer).
Main stack operations.

EXP (x --- f) $f = 3x^2+5x+1$

: EXP DUP 3 * 5 + * 1 + ; $f = x(3x+5)+1$

Exercises

Here is a list of the FORTH words we've covered in this chapter:

CONSTANT xxx xxx:	(n --) (-- n)	Creates a constant named xxx with the value n; the word xxx returns n when executed.
80 CONSTANT ROWLEN		
VARIABLE xxx xxx:	(--) (-- adr)	Creates a variable named xxx; the word xxx returns its address when executed.
CREATE xxx xxx:	(--) (-- adr)	Creates a dictionary entry (head and code pointer only) named xxx; the word xxx returns its address when executed.
! store	(n adr --)	Stores a 32-bit number into the address.
@ fetch	(adr -- n)	Replaces the address with its contents.

A := 1 1 → A 2 A +!

+! plus-store	(n adr --)	Adds a number to the contents of the address.
ALLOT	(n --)	Adds n bytes to the parameter field of the most recently defined word.
, comma	(n --)	Compiles n into the next available cell in the dictionary.
C! c-store	(b adr --)	Stores an 8-bit value into the address.
C@ c-fetch	(adr -- b)	Fetches an 8-bit value from the address.
FILL	(adr n b --)	Fills n bytes of memory, beginning at the address, with value b.
BASE	(n --)	A variable which contains the value of the number base being used by the system.

2..37

$1991_{10} = ?_{16}$ HEX

DECIMAL

1991 16 BASE ! . ØA BASE !

Some (random) terms

Execute — to perform. Specifically, to execute a word is to perform the operations specified in the compiled definition of the word.

Extensibility — a characteristic of a computer language which allows a programmer to add new features or modify existing ones.

Glossary — a list of words defined in FORTH, showing their stack effects and an explanation of what they do, which serves as a reference for programmers.

Infix notation — the method of writing operators between the operands they affect, as in "2 + 5."

Input stream — the text to be read by the text interpreter. This may be text that you have just typed in at your terminal, or it may be text that is stored on disk.

Interpret — (when referring to FORTH's text interpreter) to read the input stream, then to find each word in the dictionary or, failing that, to convert it to a number.

LIFO — (last-in, first-out) the type of stack which FORTH uses. A can of tennis balls is a LIFO structure; the last ball you drop in is the one you must remove first.

Postfix notation — the method of writing operators after the operands they affect, as in "2 5 +" for "2 + 5." Also known as Reverse Polish Notation.

Stack — in FORTH, a region of memory which is controlled in such a way that data can be stored or removed in a last-in, first-out (LIFO) fashion.

Stack overflow — the error condition that occurs when the entire area of memory allowed for the stack is completely filled with data.

Stack underflow — the error condition that occurs when an operation expects a value on the stack, but there is no valid data on the stack.

Word — in FORTH, the name of a definition.