FORTH: a pioneering language

Lennart Benschop

T-DOSE 2026-06-06 13:00-14:00

Agenda

- History
- Language features
- Under the hood
- FORTH’s strengths
- Forth today

History

- 1969-1970: Invented by Charles Moore
- 1970s: Kitt Peak National Observatory
  - Telescope and instrument control
  - 16-bit minicomputers
- 1978: FIG Forth
  - Distributed as printed assembly language source.
  - Public domain.
  - On top of BIOS no OS, no file system, source code in blocks

1980s

- FORTH is often the first language to run on new CPU
- 1984: F83
  - MS-DOS and CP/M versions
  - Source code in OS files, but in 1 kB text blocks.
  - Can rebuild itself from within FORTH
- 1988: F-PC
  - MS-DOS can use full 640 kB.
  - Source code in normal text files
  - Normal looking assembler

1990s

- 1994: ANS Forth standard
  - Allows other than 16-bit word size
- Open Firmware
- 1996: gforth
- Win32Forth (based on F-PC).

Language Features

- No traditional parser
  - Scans space delimited words (sometimes double quote or parenthesis delimited) and executes them.
  - Reverse Polish Notation, explicit data stack.
  - One data type (machine word).
  - Compile and interpret states.
  - Control structures use explicit data stack to lay out branches and resolve addresses.

Stacks

- Two stacks: data and return stack
  - Reverse Polish Notation.
  - A ‘naked’ number is just pushed onto the stack.
  - An operator like ‘+’ pops operands off the stack and pushes result.
  - Words to manipulate data on stack ‘DUP’, ‘SWAP’, ‘DROP’, avoid variables.
  - Return stack used for call/return of compiled subroutines, also for loop context and temporary value storage.

Example

- Example of RPN expression followed by '.' to print it.

```
12 23 * 44 + .
```

- Equivalent to 12*23+44 expression in most other languages
- 12 23 and 44 are numbers, pushed on the stack when encountered.
- '*', '+' and '.' are words (space delimited) to be executed.

Compiled words

- Compiled words are called ‘colon definitions’, started with the ‘:’ character
- In compile state:
  - Address of each word is added to the compiled definition.
  - Naked numbers are compiled into Literals.
  - Some words execute even during compilation. Control structs like ‘BEGIN’..’UNTIL’ or ‘IF’..’THEN’. These are IMMEDIATE words.
- The ‘;’ word ends compilation state and adds a return instruction to the compiled word.

Example

- Example colon definition

```
: PASS-FAIL 5 > IF ." Pass" ELSE ." Fail" THEN ;
```

- The word ‘:’ creates a new colon definition.
- ‘5’ is a literal, FORTH adds a LIT primitive followed by number 5.
- ‘>’ is a regular word, execution address added to the definition.
- ‘IF’, ‘ELSE’, “THEN’, ‘.”’ and ‘;’ are immediate words. IF, ELSE, THEN add and resolve branch instructions, .” adds a handler for printing the string and embeds the following string.

Defining words

- Defining word contains CREATE, so when it is executed, it creates a new FORTH word.
- Defining word contains DOES>. Which specifies that the newly created word will do.
- Comparable to class with a constructor and a single bound method.

Example

- Create a defining word
- Code after CREATE adds number from stack to dictionary (data for the object) Code after DOES> causes each object to fetch and print that number

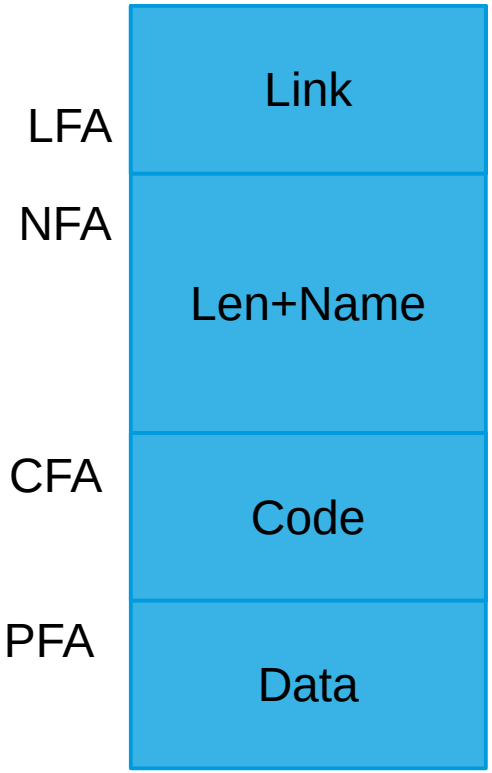
```
: FOO CREATE , DOES> @ . ;
```

- Instantiate an object using defining word (creates object named TWELVE with number 12 in it.

```
12 FOO TWELVE
```

- Use the object TWELVE (this prints number 12).

Dictionary Structure



Points to previous dictionary entry

1 byte len + Flags (IMMEDIATE)
1 byte per name char, alingned to word boundary

Points to executable machine code (contains in case of DTC)

Value for constant or variable
List of execution addresses of constituent words for colon def
Machine code for CODE words.

Under the Hood

- Threaded code
  - Compiled definition contains list of addresses of words to execute. The address can be of:
    - Primitive like “+”.
    - Compiled definition like “WORDS”.
    - Variable, constant, object created by defining word
    - Helper words like ‘LIT’ or ‘?BRANCH’.
  - Interpreter fetches next address at ‘instruction pointer’, increments instruction pointer, then executes from that address.
  - Each primitive jumps to interpreter or inlines its instructions.

Threading Techniques

- Indirect Threaded Code (traditional FORTH):
  - Execution address points to code field, code field contains address of machine code.
- Direct Threaded Code:
  - Execution address directly points to machine code.
- Token Threaded Code:
  - Compiled definition contains token values for each contining word instead of addresses, like byte code.
  - Used by some FORTH implementions in C.

Threading Techniques

- Subroutine Threaded Code.
  - Compiled definitions contain subroutine calls to the containing words.
  - No interpreter.
  - Forces the CPU ‘stack pointer’ to be used for return stack.
  - Code can be inlined, gradual path to native compiled FORTH.
- Native Code Compiler.
- FORTH CPUs (like Novix NC4000).
  - With two hardware stacks, FORTH primitives as machine instructions.

FORTH’s Strenghts.

- Low memory requirements.
  - FIG Forth with editor usable in 16 kB of RAM.
  - Less RAM required if FORTH itself lives in ROM.
- Fast edit-compile-debug cycle
- Interactive REPL.
- Some of the extensibility normally found only in LISP like languages (defining words, compiler extensions). But at a fraction of the CPU overhead.

FORTH’s strenghts

- Interactive bottom-up design of hardware control.
  - Embedded system could have an interactive REPL in FORTH.
  - Write to single ports interactively and see if the expected things happen.
  - Write low level words to access those ports, work upward for more complex actions.

Speed of FORTH

- Threaded code is slower than true compiled code.
- Faster than anything else interpreted: BASIC, p-code Pascal etc.
- Can use integrated assembler for speed critical functions
- Native compiling FORTHs exist.

Decline

- FORTH is simple but not easy.
- 1980s: IDEs on PCs, starting with Turbo Pascal. Provided short turn-around times for native PC applications.
- Early 2000s: Freely available cross compilers for embedded targets. Arduino could have been a great FORTH target.
- 2022: MicroPython. Brought the interactive REPL to microcontrollers.

FORTH today

- Implementations available for everything under the sun. including open-source ones.
  - Windows
  - Unix (including MacOS, Linux, *BSD).
  - Android
  - IOS
  - Microcontrollers (16-bit 32-bit, ESP32, ARM Cortex-M, RISC-V).
  - Forth-based operating systems (duskos https://duskos.org) on bare metal Raspberry Pi).

FORTH today

- For FORTH resources see: https://forth-standard.org
- CIForth (https://github.com/albertvanderhorst/ciforth)
  - Anything x86 (32 and 64-bit), Windows, Linux.
- Very small forth created by me, Written in C, currently only works on little-endian machines (x86, arm, riscv). https://github.com/lennart-benschop/embeddable-forth.git

FORTH today.

- GNU Forth (gforth) under Unix.
  - Full-featured, real threaded code.
  - Sadly no longer in Debian.
  - Can install from https://gforth.org
- Mecrisp (for MSP430, ARM Cortex-M and more)
  - https://mecrisp.sourceforge.net/
- Zeptoforth (for ARM Cortex-M based systems).
  - Runs on Raspberry Pi Pico, via UART or USB.
  - Get from https://github.com/tabemann/zeptoforth

Example in Zeptoforth (blink a LED)

```
variable period
500 period !
led import
: blink
  begin
    1 green led!
    period @ ms
    0 green led!
period @ ms  key? until ;
```

The End

- Any Questions?