The Left-Hand Path
dark confessions of a Forth hobbyist

Glyn Faulkner
EuroForth 2022

2022-09-17

This is the gateway to Hell, baby...
Welcome to The Underworld.
— *Kassandra Cross*

The Left-hand Path

In Western Esotericism...

Right-hand path	magic used for good, or guided by a code of ethics
Left-hand path	magic used for evil, or without consideration of morality

In Forth...

The Right-hand Path	Forth used as a powerful tool to solve real-world problems quickly and efficiently
The Left-hand Path	writing many Forth and Forth-adjacent language interpreters that the world *definitely* doesn't need

Joy: the Gateway Drug

The original concatenative functional language by Manfred von Thun

```
calc ==
  [ numerical ]
  [ ]
  [ unswons
    [ dup [+ - * /] in ]
    [ [ [calc] map uncons first ] dip call ]
    [ "bad operator\n" put ]
    ifte ]
  ifte;
```

From Joy to Funeral

A concatenative functional language in Polish notation

```
def reverse [ fold 'cons [] ]
def odd [= 1 % 2]
def even [= 0 % 2]

-- fold [fun] start [list]
def *fold [ fold dig 2 'dup ]
def fold [
    doif
        [ dip 'drop drop ]
        [ fold *apply dip 'swap dig 2 'uncons ]
    dip 'unrot *null rot
]
def rot  [ exhume 2 ]
def unrot [ bury 2 ]
```

Funeral (2011-12)

Used for HTML generation

```
html body div h1 "Hello World"

def html    [ newTag "html" setDefault
                xmlns="http://www.w3.org/1999/xhtml" ]
def body    [ newTag "body" ]
def div     [ newTag "div" ]
def h1      [ newInlineTag "h1" ]
```

including some ugly proprietary markup needed for a work project

```
def guess_value_from_name [
    doif
        [prepend "<<IPF~" append ">>" drop]
        [prepend "<<" append cons .. append ">>" ]
    = "IPQ" dup take 3 dup
]
```

Cantilever (2014)

An indirect-threaded Forth-like written in 32-bit x86 assembly, inspired by JonesForth with influences from Joy and Funeral

```
: --   [ '\n' = ] scan-in   ;   #immediate
-- and now we have comments. Yay!
```

First class dates

```
 0,1 ok 2022-02-28  1 + putdate  nl
        2020-02-28  1 + putdate ;
2022-03-01
2020-02-29
```

And times

```
 0,1 ok 11:33:30  32 +  puttime ;
11:34:02
```

The downward spiral

HackForth (2014)

```
word NextWord 8 "nex"   # ( -- label )
    call SkipSpaces
    call MakeLabel
    addl %ecx, %edx
    movl %edx, (next_input)
end
```

Thing (2014)

```
prim compile_lit ",lit"  # ( n -- )
    m_dup
    movl $lit, %eax
    call compile_call
    stosl
    m_drop
ret
```

The Quest for Minimalism

STTW (2015)

```
op fetch  "@"      _dup ; mov (%edx), %eax
op store  "!"      mov %eax, (%edx) ; _drop
```

TinyASM (2018)

```
( ?0 w1 w2 ...  if x is non-zero skip w1 )
: ?0  ( x -- ) 0<> cell-size and >r + r> ;
```

FifthWheel (2018)

```
?: dup ( n -- n n ) dsp@ @ ;
?: drop ( x -- ) dsp@ cell+ dsp! ;
```

Rage-coding

Projects I started due to anger or frustration, then quickly abandoned once I had calmed down.

WebOfHate (2018)

A small memory-footprint web browser that puts the user, instead of coporations, back in control (reaction to trying to compile Chromium from source)

BootstrapFromMBR (2020)

Let's throw our operating systems away and return to the stone-age (reaction to *all* modern operating systems!)

Wide (2021)

A tiny Forth IDE, intended to include compiler, debugging tools and full-featured editor (reaction to learning that the Atom editor exists)

The Need for Speed (of development)

OneDayProject (2022-03-08)

A native code compiler in approximately sixty x86 machine instructions.

```
despatch: # read a 16-bit token and despatch
          # based on the two high-bits.
    _dup
    xor %eax, %eax
    mov _src, %esi
    subl $1, _slen
    jc bye
    lodsw
    mov %esi, _src
    xor %ecx, %ecx
    shld $2, %ax, %cx
    shl $2, %ax
    mov handlers(,%ecx,4), %ecx
    jmp *%ecx
```

Lessons learned walking the left-hand path

- Replacing `lods` with a separate move from memory and add is often a performance gain for ITC code
- Replacing `lods` with `pop` also works (bigger difference on Intel)!
- You *can* implement direct-threading using `ret` as `NEXT` and ESP as the instruction-pointer. But just don't.
- Binary source code generally isn't a great idea. . .
- . . . *but* having a binary-token intermediate representation simplifies your compiler and speeds the process of bootstrapping a new Forth.
- It does not appear to be possible to fit a useful Forth system into the 510 bytes available on an x86 boot sector.
- You can write an assembler in Forth using only `c,...` if you enjoy pain.
- Forth can be bootstrapped using a subset of Forth, without the need for compile-time execution, as `if` and `recurse` represent predictable sequences of instructions.
- It is possible to bring-up a rudimentary Forth system in one day, even in assembly.
- GNU assembler isn't as bad as you think.
- Rage-programming is rarely productive!

Where next?

How about a parameterised Forth interpreter generator?

```
[marsu@celaeno 4g]$ ./4g -t ITC -T -m ANSI -o forth
Indirect-threaded x86_64 Linux ANSI Forth
Options: top-of-stack in register, linked-list dictionary
Generating forth.S
gcc -m64 forth.S -o forth
Done
[marsu@celaeno 4g]$ ./forth
```

Ask me how this is going next year!

Comments/Questions?