Implementing a Forth

- Forth Background.
- Why a Forth?
- Stack Machines.
- Implementation Concepts.
- Execution & Threading.
- Stacks Operations & Postfix.
- *epop* Overview & Examples.
- Resources.

Jack.Pope@ieee.org

- Investment data science infrastructure on UNIX, since ancient times.
- Chairman, Twin Cities IEEE Computer Society.
- Computer Science / Data Science, Minnesota State Colleges & Universities.
- Developer of *epop*, a Forth inspired programming environment.

Disclaimer

Programming Languages may have standards

(IEEE POSIX, ANSI Forth, ANSI C, ...)

Compiler implementations have principles.

(No rigid rules.)

Forth History

Developed by Charles Moore in the late 1960’s.

- A student of John McCarthy at MIT in the 1950s.
- Possibly influenced by McCarthy’s LISP programming ideas.
- Forth was “functional” long before there was *Functional Programming*.

Why Forth?

- The efficiencies of a stack machine.
- Can be self-hosted and be its own OS.
- A compiler-implementation paradigm.
- More than a Programming Language.

Why Forth, cont’

- A problem solving language:
  - Compact / concise expressions.
  - Self-documenting syntax.
  - *Factoring* words:
    - Identify general problem / solution.
    - Identify most basic component words.
    - Inductively compose solution of component words.

Host vs Guest System

- Hosted: Host language defines Forth dictionary.
- Self-hosted: Guest system language defines Forth dictionary (minimal machine level / assembly functions).

Minimal Host Components

- Push function: for data stack
- Pop function: for data stack
- Data Stack
- Program Stack

Stack Machines

- Stack: Dedicated registers or dedicated area of memory.
- Stack data is Last-In-First-Out (LIFO).
- Program Stack: instruction sequence.
- Dictionary: A parallel in-memory structure/table.
- Stack Counter: element count; size-of.
- Stack Pointer (top of the Program Stack):
  - Memory address of next instruction.
  - May be indexed by Program Counter.
- Push data (to top of Data Stack).
- Pop data (from top of Data Stack).
- Return Stack:
  - Addresses of functions that call other functions (return address) for continuing program sequence.
  - And/or auxiliary data stack for the current operation.

Forth Execution

- Compile-time generation of host language functions.
- Compile-time generation of guest language functions. (Like Forth’s CREATE DOES> sequence)
- Run-time Virtual Machine: loop -> word parse / tokenize -> stack(s) -> exec

The Virtual Machine Loop

- Read text input -- via user interface or file i/o.
- Interpret / parse -- one or two passes with look-ahead tokenizer.
- Generate high level program (abstract word tree).
- Recursively flatten tree to low level program stack.
- Evaluate program stack.
- Repeat

Indirect Threaded Code

- Portable: No predefined function addresses (not direct).
- More low-level jumps than direct threaded code.
- Replace words (abstract functions) with:
  - Primitive addresses
  - Intermediate opcode
  - Intermediate abstract object (token or subroutine threading)
- Dispatch the replacements to program stack.

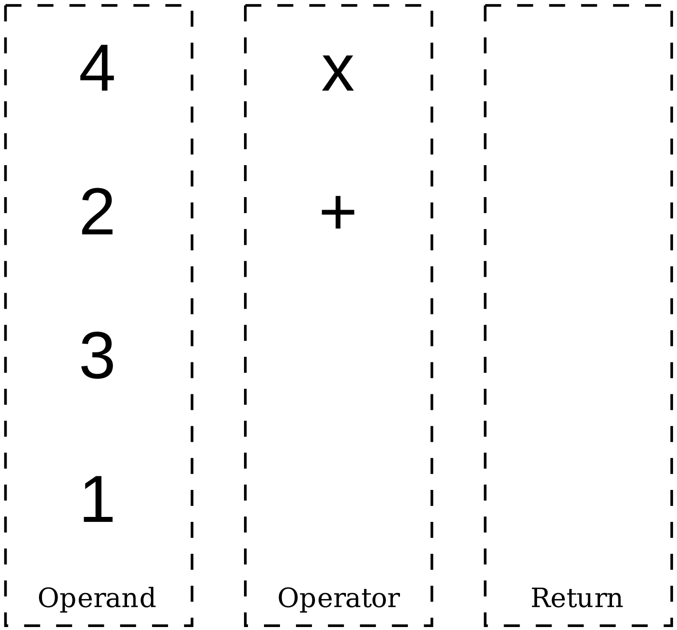
Word Dispatch: Vectored Execution

- Replace input vector of abstract words with by executable objects.
- Flatten abstract word tree into executable program stack (indirect threading).
  - Use recursive descent operations with
  - Switch statement (switch threading).
- Identify next word (opcode, address,...).
  - IF condition is 1 “immediate” then exec.
  - ELSE push word to program stack.
- Advance stack pointer/counter.
- Execute the program stack.

Stack Operations: Postfix

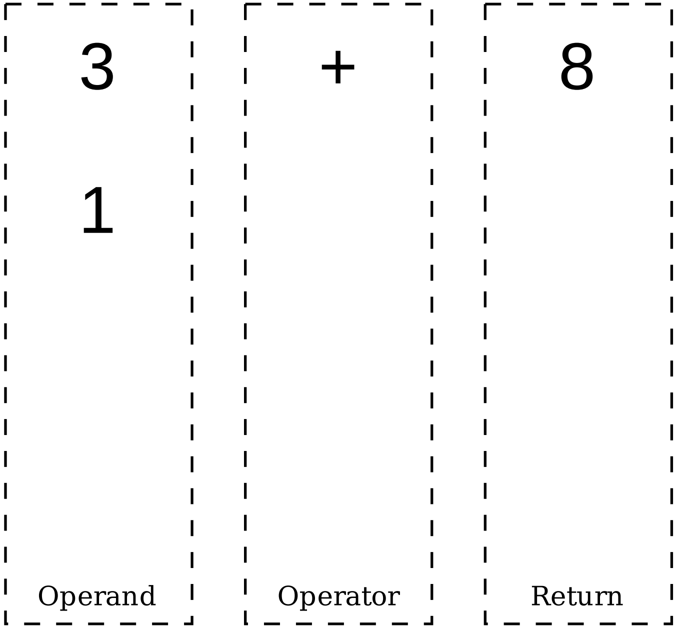
- Efficient for memory and CPU.
- No rules of precedence.
  - No need of ( ) parentheses, unlike infix notation.
- Linear processing from left-to-right; top-to-bottom.
- Think of assembly’s prefixed notation, in reverse.
- Ex: 2 1 + --> 3

Separate Stacks



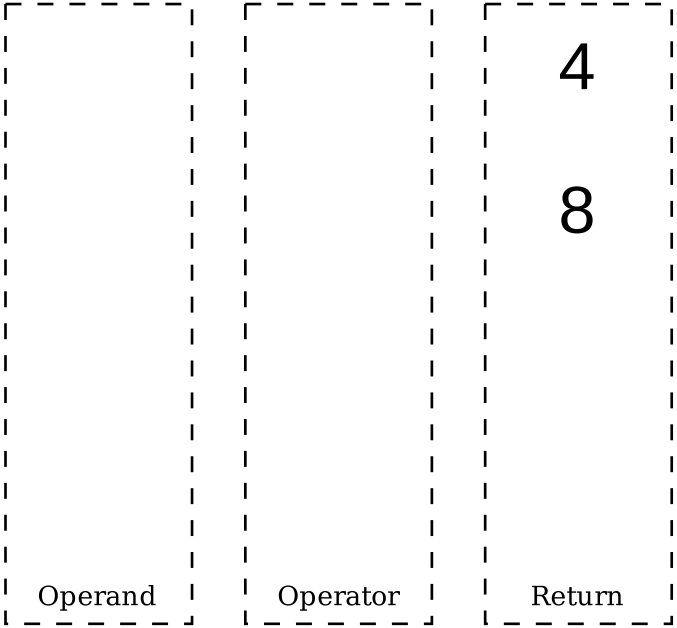
Three Separate Stacks (before operations).

Separate Stacks



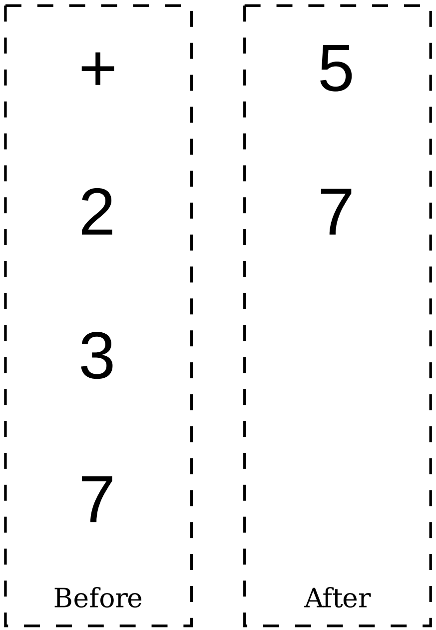
Three Separate Stacks (after one operation).

Separate Stacks



Three Separate Stacks (after two operations).

Combined Stacks



A Combined Stack (before & after one operation).

epop overview

- Most operations are postfix and stack oriented.
- Program stack: Linear linked-list.
- Three data stacks: Circular linked-lists (for memory management).
- Data types: numeric, string, table and XT.
  - Use tables for “big data.”
  - XTs (execution tokens) can be treated as data.
- Run-time user defined words (in REPL).
- Compile-time “Forth” word definitions and programs.
- Compile-time primitives (D & C host code).
  - D APIs for SQLite (RDBMS) and Curl (networking).

Example: Put data on stack

```
epop> 1 3 5 7 9
epop> .S
Dat stack:
Value   index     address
---------------------------------
  9       4       82A163630
  7       3       82A163580
  5       2       82A165E70
  3       1       82A165C60
  1       0       82A1659A0

epop> 
```

Example: Data stack as “program”

```
epop>
epop> 1 2 ' +
epop> .S
Dat stack:
Value   index     address
---------------------------------
  ADD     2       829E05420
  2       1       829E03630
  1       0       829E03580

epop> EXEC . CR
3
epop>
```

Example: Sum data on stack

```
epop> ODDS
epop> .S
Dat stack:
Value   index     address
---------------------------------
  9       4       82B1869A0
  7       3       82B186840
  5       2       82B186790
  3       1       82B1866E0
  1       0       82B186630

epop> { + } GSC i- LREPEAT
epop> .S
Dat stack:
Value   index     address
---------------------------------
  25      0       82B174420

epop>
```

Example: Define word to sum data

```
epop>
epop> : SUMDAT { + } GSC i- LREPEAT ;
epop> ODDS
epop> SUMDAT . CR
25
epop>
```

Example: Define factorial as map & reduce operations

```
epop> : MAP DUP IF DUP >R i- RECUR ELSE THEN ;
epop> : REDUCE GRC IF R> * RECUR ELSE THEN ;
epop> : FACT MAP REDUCE ;
epop> 5 { FACT . } LE
120
```

Example Program: ASCII printout

```
( ascii.epop )
( print out ascii table )

: tab 9 EMIT ;
: hdr chr . tab dec . tab bin . tab tab oct . tab tab hex . CR ;
: ascii_row   DUP 0 BASE . tab   DUP 10 BASE . tab   DUP 2 BASE . tab   DUP 8 BASE . tab   DUP 16 BASE tab .   CR ;

hdr
64 { i+ ascii_row DUP } 26 LREPEAT
```

About ASCII:
https://www.w3schools.com/charsets/ref_html_ascii.asp

Example: Run the ASCII program

```
epop>
epop> ascii 1 RUN
chr     dec     bin             oct             hex
A       65      b1000001        101             h41
B       66      b1000010        102             h42
C       67      b1000011        103             h43
D       68      b1000100        104             h44
E       69      b1000101        105             h45
F       70      b1000110        106             h46
G       71      b1000111        107             h47
H       72      b1001000        110             h48
I       73      b1001001        111             h49
J       74      b1001010        112             h4A
K       75      b1001011        113             h4B
L       76      b1001100        114             h4C
M       77      b1001101        115             h4D
N       78      b1001110        116             h4E
O       79      b1001111        117             h4F
P       80      b1010000        120             h50
Q       81      b1010001        121             h51
R       82      b1010010        122             h52
S       83      b1010011        123             h53
T       84      b1010100        124             h54
U       85      b1010101        125             h55
V       86      b1010110        126             h56
W       87      b1010111        127             h57
X       88      b1011000        130             h58
Y       89      b1011001        131             h59
Z       90      b1011010        132             h5A
epop>
```

Example Program: Fetch HTTP data

```
( dog-is-dog.epop )

CrAzYpAsSwOrD pssw !
SomeUser unam !
datamart.systemgoats.com/a-dog-is-a-dog.txt url !

url @
unam @
pssw @

HTTPGET
.
```

Example: Run HTTP Fetch

```
epop>
epop> dog-is-dog 1 RUN CR
A dog is A Dog
by T. S. Eliot

Now dogs pretend they like to fight;
They often bark, more seldom bite;
But yet a Dog is, on the whole,
What you would call a simple soul.
Of course I'm not including Pekes,
And such fantastic canine freaks.
The usual Dog about the Town
Is much inclined to play the clown
And far from showing too much pride
Is frequently undignified.
He's very easily taken in-
Just chuck him underneath the chin
Or slap his back or shake his paw,
And he will gambol and guffaw.
He's such an easy-going lout,
He'll answer any hail or shout.

Again I must remind you that
A Dog's a Dog - A CAT'S A CAT.
```

Example: See user defined words

```
epop>
epop> u WORDS

Dictionary (symbol table):
---------------------------------------------------
 opcode      symbol       type      CW/members
---------------------------------------------------
   237       Z            u         0.0 D
   238       ONE          u         1 D
   239       -ONE         u         -1 D
   240       ONES         u         1 1 1 1 1 1 1 1 D
   241       ZEROS        u         0 0 0 0 0 0 0 0 D
   242       ODDS         u         1 3 5 7 9 D
   243       EVENS        u         2 4 6 8 10 D
   244       NULL         u         00 D EMIT
   245       BS           u         8 D EMIT
   246       CR           u         10 D EMIT
   247       SPACE        u         32 D EMIT
   248       TRUE         u         1 D
   249       FALSE        u         0 D
   250       i+           u         1 D ADD
   251       i-           u         1 D SUBTRACT
   252       i*           u         1 D MULTIPLY
   253       Z=           u         0.0 D EQ
   254       Z>           u         0.0 D GT
   255       Z<           u         0.0 D LT
   256       NIP          u         SWAP DROP
   257       NEGATE       u         -1 D MULTIPLY
   258       ROT          u         3 D MOVE
   259       -ROT         u         3 D MOVE 3 D MOVE
   260       OVER         u         2 D PICK
   261       DUP2         u         2 D PICK 2 D PICK
   262       TUCK         u         SWAP 2 D PICK
   263       RUP          u         SWAP 2 D PICK SUBTRACT ADD SWAP
   264       OR           u         ADD 0.0 D GT
   265       NOT          u         IF 0 D ELSE 1 D THEN
   266       AND          u         MULTIPLY 0.0 D GT
   267       XOR          u         0.0 D GT SWAP 0.0 D GT ADD 1 D EQ
   268       ABS          u         DUP 0.0 D LT IF -1 D MULTIPLY ELSE THEN
   269       SQUARE       u         DUP MULTIPLY
   270       NEQ          u         EQ IF 0 D ELSE 1 D THEN
   271       R@           u         PULLR DUP PUSHR
   272       RDROP        u         PULLR DROP
   273       SUMDAT       u         BLE ADD ELE GSC 1 D SUBTRACT LREPEAT
epop>
epop>
```

Example Program: Run from CLI

```
fbsdev ~/epop:
fbsdev ~/epop: epop RUN $EPOP_HOME/APPS/hello.epop

Hello_World!

fbsdev ~/epop:
```

Related Resources on the Internet

- https://forth-standard.org
- https://www.forth.com/starting-forth
- http://forth.org/compilers.html
- http://www.bradrodriguez.com/papers/moving1.htm
- https://en.wikibooks.org/wiki/Compiler_Construction
- https://users.ece.cmu.edu/~koopman/stack_computers
- http://www.complang.tuwien.ac.at/forth/threaded-code.html
- https://compilers.iecc.com/crenshaw
- https://openfirmware.info/Bindings

Forth Systems

- Forth Systems: https://forth-standard.org/systems
- Compilers written in Forth:
  - https://bellard.org/tcc/
  - https://arduino-forth.com/article/FORTH_metacompilation_intro
  - https://git.sr.ht/~vdupras/duskos/tree/master/item/fs/comp/c
  - https://www.mpeforth.com/arena/C2ForthKit.120.zip
  - https://github.com/pzembrod/cc64

https://systemgoats.com/epop.html