Introduction to Forth for Scientists and Engineers

January 2004

NASA/CXC/SAO

- History of Forth
- Overview of Forth
- Applications of Forth
- Forth Language
- Forth Example
- Why Use Forth?
- Forth Systems
- System Benchmarks
- Forth Resources

History of Forth

- Forth was developed by Chuck Moore in the 1960s (see Forth - The Early Years by C. Moore and The Evolution of Forth by E. Rather, et al).
- Original use for Forth was to perform instrument control, data acquisition, and least-squares curve-fitting at NRAO and Kitt Peak.
- Became a formal programming language in 1977 with Forth-77 standard. Subsequent standards were Forth-79 and Forth-83 by the Forth Standards Team.
- First commercial Forth system for IBM-PC introduced in 1982 by Laboratory Microsystems, Inc.
- Became an ANSI standard language in 1994, resulting in ANS-Forth.

Overview of Forth

- Forth is a computing environment.
  - Forth is interactive.
    - Perform computations directly at the Forth prompt.
    - Define and examine variables and constants
    - Define and execute new Forth words (individual subroutines).
    - Execute operating system commands.

ok

Overview of Forth

- Forth is a high-level language.
  - Forth is structured:
    - Like all modern programming languages, Forth provides the necessary control structures for prescribing an ordered flow of execution.
  - Forth is extendable:
    - Forth provides no built-in support for arrays, data structures, lists, objects, etc., but Forth allows the user to add such programming constructs to the language itself.
    - Forth can support all programming methods: procedural, modular, object-oriented, or whatever new comes along …

ok

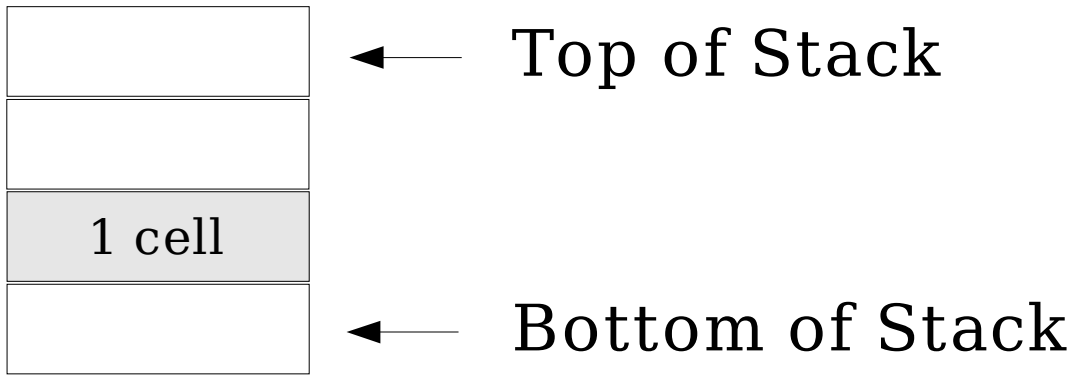
Overview of Forth

- Forth is a low-level language.
  - Forth provides bit-level operations and selection of number base.
    - The user can work directly in hex and binary bases --- the language of hardware. Results of single hardware operations, such as writing to a port, can be verified immediately.
  - Most Forth systems provide an assembler
    - The assembler is often a Forth program itself. The programmer can switch back and forth between Forth and assembly code within the same program.

ok

Overview of Forth

- Forth syntax is derived from use of a data stack.
  - The basic method of passing arguments to, and obtaining results from, Forth words is through the data stack.



ok

Overview of Forth

- Forth maintains a list of words, or a **dictionary**.

```
words
WORD          WORDS         FIND          '             [']
[             ]             CREATE        DOES>         >BODY
FORGET        COLD          ALLOT         ?ALLOT        LITERAL
EVALUATE      IMMEDIATE     CONSTANT      FCONSTANT     VARIABLE
FVARIABLE     CELLS         CELL+         CHAR+         DFLOATS
DFLOAT+       SFLOATS       SFLOAT+       ?             @
!             2@            2!            A@            C@
C!            W@            W!            F@            F!
DF@           DF!           SF@           SF!           SP@
RP@           >R            R>            R@            2>R
2R>           2R@           ?DUP          DUP           DROP
SWAP          OVER          ROT           -ROT          NIP
TUCK          PICK          ROLL          2DUP          2DROP
2SWAP         2OVER         2ROT          DEPTH         BASE
BINARY        DECIMAL       HEX           1+            1-
2+            2-            2*            2/            DO
?DO           LOOP          +LOOP         LEAVE         UNLOOP
I             J             BEGIN         WHILE         REPEAT
UNTIL         AGAIN         IF            ELSE          THEN
CASE          ENDCASE       OF            ENDOF         RECURSE
BYE           EXIT          QUIT          ABORT         ABORT"     ...
```

ok

Applications of Forth

- Embedded Systems:
  - smart cards, robotics, Fed-Ex package trackers, embedded web servers, space applications
- Software Tools Development
  - writing cross-assemblers and disassemblers
  - writing parsers and programming languages
  - scripting and software testing
- Application Development
  - editors, word processors, games, circuit modeling, VLSI design, ...
- Laboratory Automation
  - Hardware Interfacing
  - Data acquisiton, data logging
  - Instrument control
- Engineering and Scientific Computing
  - Data analysis
  - Simulation and modeling
  - Visualization
- Exploratory Computing
  - algorithm development
  - artificial intelligence programming, cellular automata, evolutionary programming

Forth Language

Stack Operations:

```
DUP   SWAP   ROT    DROP   OVER
>R    R>     ?DUP   NIP    TUCK
PICK  .S     .      2DUP   ...
```

Examples:

```
                    2
1 2 .S              1

                    1
1 2 SWAP .S         2

                    1
                    3
1 2 3 ROT .S        2
```

Forth Language

Integer Arithmetic:

```
+     -     *        /     */
MOD   /MOD  1+       1-
NEGATE      ABS
```

Examples:

```
3 8 * . 24 ok

56 5 MOD . 1 ok

1048576 10120  153 */  .  69356791 ok
```

Forth Language

Relational Operators:

```
 =   <    >    <=    >=
0=   0<   ...
```

Examples:

```
1 3 < . -1 ok

4 0= .  0 ok

-5 -2 <= .  -1 ok
```

Forth Language

Bitwise Operators:

```
AND     OR      XOR   INVERT
LSHIFT  RSHIFT  2*    2/
```

Example:

```
: byte-swap ( n – m )
    DUP 8 RSHIFT SWAP 255 AND 8 LSHIFT OR ;

4096 byte-swap .  16 ok
```

Forth Language

Number Base Operations: `DECIMAL  HEX  BASE`

Example:

```
HEX  FFD2  DECIMAL . 65490 ok

2 BASE !
10111001  DECIMAL . 185 ok

-1
2 BASE ! U.  11111111111111111111111111111111 ok
```

Forth Language

Branching:

```
IF ... THEN
IF ... ELSE ... THEN
CASE ... OF ... ENDOF ... ENDCASE
```

Example:

```
: even?  ( n -- )
    2 MOD 0= IF ." YES" ELSE ." NO" THEN ;

5 even?  NO ok
8 even?  YES ok
```

Forth Language

Looping:

```
DO ... LOOP        ?DO ... LOOP
DO ... +LOOP       ?DO ... +LOOP
I  J
BEGIN ... AGAIN
BEGIN ... UNTIL
BEGIN ... WHILE ... REPEAT
```

Example:

```
: 2^  ( n – 2^n)  1 SWAP LSHIFT ;

: pow2-sum ( n – m | sum of terms 2^i, i=0,n-1)
   0 SWAP 0 ?DO  i 2^ + LOOP ;

 10 pow2-sum .  1023 ok
```

Forth Language

Indefinite Loop Example:

```
: pad2 ( n – m | m is next power of 2, >= n)
   DUP 0 <= IF DROP 1 THEN 1
   BEGIN
     2DUP >
   WHILE
     2*
  REPEAT
  NIP ;

348 pad2 . 512 ok
```

Forth Language

Recursion Example:

```
\ Find the greatest common divisor of two
\ integers

: gcd  ( n1 n2 -- gcd )
    ?DUP IF SWAP OVER MOD RECURSE THEN ;

1050 432 gcd .  6 ok
```

From A Beginner's Guide to Forth by J.V. Noble

Forth Example

Forth Example

Timing is everything ...
some basic words we will use:

```
: launch-clock@ ( -- t | return a signed time in 1/100th of seconds)
     ( ... system dependent code ... ) ;

: hms>s ( h m s – s2 | convert hours, min, sec to seconds)
     >R 60 * >R 3600 * R> + R> + ;

: hmsh>t ( h m s hs – t | convert to hundredths of seconds)
   >R hms>s 100 * R> + ;

: T- ( h m s hs – t )  hmsh>t NEGATE ;
: T+                   hmsh>t ;

: is-time? ( t – flag | is t <= launch clock ?)  launch-clock@ <= ;
```

Forth Example

```
: launch  ( -- | launch the space shuttle )
      BEGIN
      ( H  M  S  HS )
        00 00 06 60  T-  is-time?  IF  start-main-engine-3  THEN
        00 00 06 48  T-  is-time?  IF  start-main-engine-2  THEN
        00 00 06 36  T-  is-time?  IF  start-main-engine-1  THEN
        00 00 00 00  T+  is-time?  IF  ignite-SRBS  release-SRBS THEN
        00 00 00 01  T+  is-time?
      UNTIL ;
```

Forth Example

Our Forth definition of **launch** is more readable than the following equivalent **C** function:

```
void launch()
{
      do
      {
        if (is-time(Tminus(00, 00, 06, 60))) start-main-engine-3();
        if (is-time(Tminus(00, 00, 06, 48))) start-main-engine-2();
        if (is-time(Tminus(00, 00, 06, 36))) start-main-engine-1();
        if (is-time( Tplus(00, 00, 00, 00)))
        {
          ignite-SRBS(); release-SRBS();
        }
      } while (! is-time(Tplus(00, 00, 00, 01)) ;
}
```

Forth Example

Forth's extendability allows us to write **launch** even more simply!

```
: launch ( -- | launch the space shuttle )
    BEGIN
     ( H  M  S  HS )
       00 00 06 60  T-  at  start-main-engine-3
       00 00 06 48  T-  at  start-main-engine-2
       00 00 06 36  T-  at  start-main-engine-1
       00 00 00 00  T+  at  ignite-SRBS  release-SRBS
       00 00 00 01  T+  is-time?
    UNTIL ;
```

Advanced Forth

However, simplicity is not free ...

```
: at ( t <"..."> -  | if t <= launch clock take actions )
     POSTPONE launch-clock@  POSTPONE <=  POSTPONE IF
     BEGIN
       BL WORD DUP COUNT NIP
     WHILE
       FIND
       IF
         POSTPONE LITERAL  POSTPONE EXECUTE
       ELSE
         DROP
       THEN
     REPEAT
     DROP POSTPONE THEN ; IMMEDIATE
```

Advanced Forth

- A word written in Forth can act as a compiler.
- "**at**" is an IMMEDIATE word.
  - When the word "**at**" is used inside the definition of a word, it compiles into the current definition the specified operations and IF ... THEN logic structure.
  - "**at**" parses the rest of the input line to place the specified actions within the IF ... THEN structure.
- In Forth we can also write a word which may be used to CREATE new words.

Why Use Forth?

- Forth allows both low-level and high-level programming.
  - A wide range of software application needs can be addressed by Forth, from writing time-critical embedded processor code to developing entirely new programming languages. Forth is ideally suited for mid-level applications such as laboratory data-acquisition and instrument control.

Why Use Forth?

- Forth simplifies testing of code at every stage of development.
  - With its interactive environment and incremental compilation, new Forth words can be tested as they are written. The bottom-up approach of building new words upon previously tested words leads to very robust code.

Why Use Forth?

- Forth can be extended to suit the application.
  - Definitions of high-level words in a well-written Forth application are simple and readable. Often, they read like a plain English description of the actions being implemented.
  - Source code which is simple and readable is less prone to programming mistakes, easier to maintain, and is self-documenting.

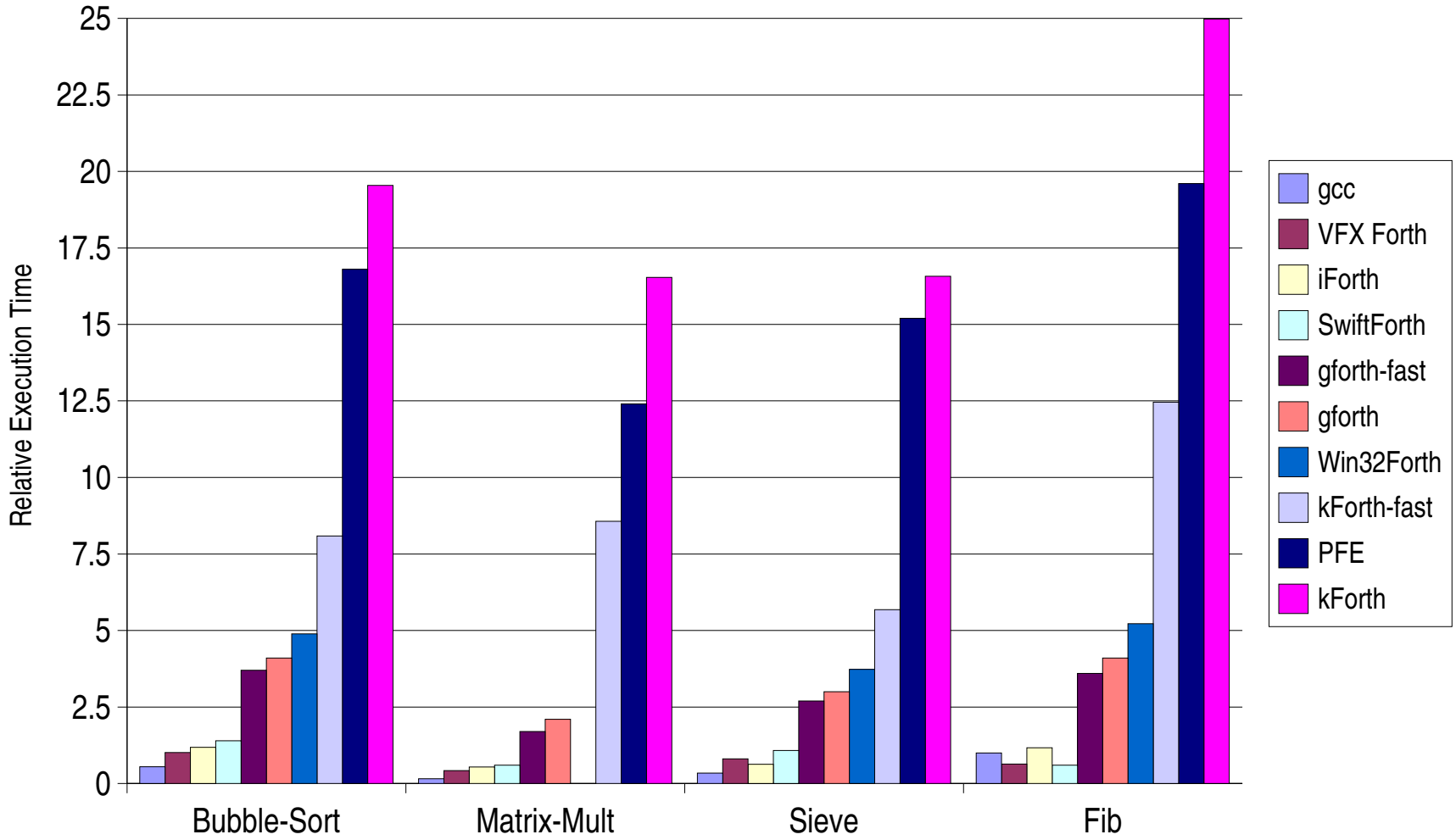
Why Use Forth?

- Forth is easy to learn.
  - Forth's interactive environment provides a quick way to test simple Forth words. The user gets instant feedback without the edit-compile-execute cycle of other languages.
  - New Forth programmers can be productive in a matter of days with the guidance of a Forth expert.

Forth Systems

- SwiftForth and SwiftX http://www.forth.com/ (Windows, embedded targets)
- VFX Forth and Cross Compilers, http://www.mpeltd.demon.co.uk/ (Windows, embedded targets)
- iForth (Windows, Linux)
- Camel Forth for embedded processors (8051, 8086, Z80, and 6809)
- gforth (DOS, Windows, OS/2, MacOS X, Unix, Linux, other)
- PFE (DOS, Windows, OS/2, MacOS X, Unix, Linux, other)
- kForth (Windows, Linux, FreeBSD, BeOS)
- Win32Forth (Windows)
- See also Other Free Forths

Forth System Benchmarks



Forth Resources

- Forth Programmers Handbook
- Forth Code Index
- comp.lang.forth
- Forth Interest Groups:
  - FIG-UK
  - FIG-USA
  - other FIGs