Star Forth

Starting Forth is available online at Starting Forth Online Edition.

Another book: Threaded Interpretive Languages.

Welcome to Star Forth

SD/FORTH 1.0

- **32-bit cells, subroutine-threaded**
- 32-bit signed integer math
- 87 native words: BYE, ., .S, CR, DUP, DROP, SWAP, OVER, +, -, *, /, MOD, EMIT, DEPTH, WORDS, ...
- 26 defined words: MIN, MAX, ABS, NEGATE, RAND, SPACES, TYPE, ROT, NIP, TUCK, ...
- Outer interpreter with TIB (Terminal Input Buffer), word parser, dictionary search, number parsing
- Control flow (IF/THEN/ELSE, BEGIN/UNTIL, DO/LOOP) and Memory words (pointers, peek, poke)
- Stack primitives with underflow protection
- Optional binary and hexadecimal input and output
- : and ; (colon definitions)
- Paste-friendly '> ' stripping
- AND, OR, XOR, NOT bitwise operations
- to-r, r-from, and r-at return stack manipulators
- VARIABLE, CONSTANT and CREATE
- EXECUTE and ' (tick) – find and run word from address
- Native U. (unsigned decimal 32 bit number, vs. HEX mode which is unsigned already)

“That's all well and good,” you may be wondering, “but can it run FizzBuzz?” Yes, yes it can; see Star Forth Test Suite

Star Forth has some words from Forth-83, such as LSHIFT and RSHIFT, and is missing some from FORTH-79. That's because you can define the remaining words inside forth itself and I think that's cool. Therefore, at start, more than 30 new words are autoloaded into the dictionary; they're defined in FORTH itself and are not native. Included are great new words like RAND (100 1 RAND gives a number from 1 to 100) and ENSEED which adds entropy to the RNG.

For a comprehensive reference to each of these commands see Star Forth Dictionary

Why Forth?

Forth is one of the most fascinating and misunderstood languages ever created. Invented by **Charles “Chuck” Moore** throughout the 60s as he worked on various machines (including the IBM 1130, the Burroughs 5500, and the radio telescopes at Kitt Peak NRAO), by the 1970s it had evolved into the language we consider Forth today. It’s a **stack-oriented, concatenative, postfix** language that feels like a mix of assembly, BASIC, REPL, a compiler, an operating system, and a philosophy all in

one.

Everything in Forth is a “word” (a subroutine). You define new words that build on old ones, and the language is extremely extensible; you can even change how the compiler works from inside the language itself. It runs in a tiny footprint (many classic implementations fit in 8 to 16 KB total) and gives you immediate, interactive feedback.

Forth vs. BASIC

Both are early 80s microcomputer languages, so the choice of BASIC Forth for a retro fantasy console is an important decision. Let's approach this from practical terms. BASIC is a requirement. But Forth? Forth was popular and was considered a very fast and useful language on computers like the C64.

BASIC	POKE 49152, 10
Forth	POKE 49152 10

This is a pretty accurate statement.

I know, I know, people are going to say, “That's not Forth!” but it is. Yes, normally in Forth you type `10 49152 POKE`, but look:

```
: POKE WORD NUMBER WORD NUMBER SWAP C! ;
: PEEK WORD NUMBER C@ ;
```

Now it just works.

50x faster than BASIC

Also, to put it bluntly, Forth destroys classic interpreted BASIC. On 8-bit computers (like the Commodore 64, Apple II, etc.) Forth was legendary. Often 20x faster than BASIC for the same tasks. Real benchmarks from the era and modern recreations show Forth loops, math, and hardware control running at near-assembly speeds. In fact, DurexForth can run programs 50x faster than BASIC for the same tasks. User response to Forth on early computers like the C64 was fantastic:

- “Fast and fun!”
- “Powerful language for an 8-bitter.”
- “Impressive for being on a C64” (e.g., the built-in editor).

Forth was mind-blowingly quick for hardware interaction (toggling sprites or drawing). Some preferred it over BASIC for expressiveness – BASIC lacked good abstractions for C64 hardware, making Forth cleaner for games and tools. It even saw use in schools – some teaching Forth alongside Logo and Pascal.

Overall, early adopters loved Forth on the C64 for its power and immediacy on a budget machine, viewing it as a step up from BASIC for those willing to learn its quirky style. It wasn't a mass-market success, but it earned a dedicated following in the scene of the time.

Forth vs. Python

Forth	10 49152 POKE
Python	You can't do this

That's a pretty accurate statement.

Ok, you can do this:

```
import ctypes
ctypes.c_uint8.from_address(49152).value = 10
```

But now you're really just programming in C.

The other problem is that Python is usually implemented as a userland program, and modern operating systems actively prevent user programs from writing to arbitrary memory. The OS will need you to grant special permissions to your python script in order to allow them to do this.

The other problem is that Python itself is exceedingly *slow*. It's even slower than BASIC. It's even slower than COMMODORE BASIC V2.0, where filling the screen with characters can take up to 10 seconds. If Python could even fit in RAM, which it can't. Python is just not on the level.

I'm not saying Python is bad. I like Python, once I learned it. But there is no contest here – Forth destroys Python in ways not even I can understand. If you're writing stuff in Python, you're taking advantage of how fast computers have become. I don't know if that is good or bad. Python is great. Go learn Python. No, really. Forth will be here for when you need it.

```
import ctypes ctypes.c_uint8.from_address(49152).value = 10
```

Forth vs. C

C	poke(49152, 10)
Forth	10 49152 POKE

That's a pretty accurate statement.

The fact that there are more ways to do this in C is not relevant. It's actually a downside of C; too much syntax that gets optimized away anyways. Or, use it if you want. Both C and Forth are highly optimized, function calling languages. In fact, most C implementations use a stack frame to pass values and hold local variables, and this is exactly how Forth works. It's just that C hides this from you and Forth exposes it.

In C, or most programming languages, the statement

```
add (a, b)
```

does nothing. That's because the stack is “reset” every line. On the other hand,

```
c = add (a, b)           // Now the return value has somewhere to go
```

This “return value” is the value a function “returns”, or pushes to the top of the stack; but having

nowhere to go, C discards it. Forth keeps it on the stack So if you do

```
add (a, b)
add (a, b)
add
```

this resolves to (a+b) plus (a+b). C can't do that. With C you would need something like:

```
c = add (a, b)
d = add (a, b)
e = c + d
```

That's the rough equivalent.

As for speed, if Forth is slower than C, it's primarily because C is a much more scrutanized language and has had far more work done on it's optimizing compilers.

Today, well-written Forth is usually within 10 to 30% of optimized C – and sometimes faster, because there’s no function-call overhead or hidden runtime.

What's more, FORTH binaries are dramatically smaller; a 10k FORTH binary will often have an equivalent C program binary of about 100k.

Is Forth better than C?

Many people say the reason C “won” over forth is because of UNIX. That's only partially true. The second major reason is because Forth is hard to learn. There, I said it.

Yet Forth is astonishingly simple and easy to understand. The entire language core can be implemented in a few hundred lines of assembly. There are no complex syntax rules, no precedence battles, no huge standard library to learn. If you make it past the first week, you will start to understand it. The key is to keep trying. As a language developed in the 60s, but polished in the 80s and 90s, it is just different than what we have become used to. Ultimately, you can redefine how the language works.

Chuck Moore’s philosophy was radical minimalism: “If the program is too big, you’re doing it wrong.” This aligns with the UNIX philosophy neatly; “Small is beautiful.” Most serious Forth programs stay tiny by modern standards because the language forces you to rethink the problem until the solution shrinks.

Forth is not a toy language. Companies and agencies paid real money for it in the 1970s, 80s and 90s because it delivered real-time performance in a small memory footprint.

It’s still actively used in niches where C is too heavy or too slow to develop: radiation-hardened space systems, FPGA soft cores, boot firmware, and low-power microcontrollers. The language has an official ANS Forth standard (1994), commercial implementations (polyFORTH, SwiftForth), and vibrant open-source ones (Gforth, Mecrisp-Stellaris). But as to whether it is better than C is a personal choice. Some people like C's style, some people like Forth's.

Many people love Forth's immediacy, minimalism, and raw power. But Forth never had an equivalent “killer app” ecosystem that pulled in masses of developers. It stayed more niche (telescope control,

industrial automation, some embedded, Open Firmware on PowerPC/SPARC).

It is ironic that today, header files are considered bad meta and many people are pushing towards less imperative languages. Perhaps if Forth had taken off more in the 70s and 80s it would have been seen as a solution to those kinds of issues and we would not have muddied the water with the so-called “modern” programming languages. There is a way out of this hole, of course – stop digging and learn Forth!

Written in FORTH

Game	Date / Publisher	About
Starflight	1986 (EA / Binary Systems)	The standout. A huge open-world space RPG with trading, combat, planet landing, and alien diplomacy. Coded mostly in Forth (plus a few x86 assembly routines for speed). It had to fit in just 128 KB of RAM, and Forth’s compactness made that possible. Sold over a million copies, won awards, and is still beloved today. Ports to Amiga, Atari ST, Mac, C64, and even Sega Genesis later.
ChipWits	1984 (BrainWorks)	A charming puzzle game where you visually program a little robot to navigate mazes and hazards. Original Mac version used MacFORTH; C64 version used SuperForth 64. The full Forth source code was released in 2024 for its 40th anniversary – perfect for retro hackers to study.
Adventure Construction Set	1984 (Electronic Arts)	One of the very first RPG/adventure game makers. You built tile-based worlds, monsters, and quests. Creator Stuart Smith confirmed in a 2020 interview that the entire thing was written in Forth on a Commodore 64.
Lords of Conquest	1986 (Electronic Arts)	Strategy/war game, also in Forth.
Amnesia	1986	Text/graphical adventure, one of the seven games officially listed in Wikipedia’s “Video games written in Forth” category.

Several official Atari 7800 console games were developed in Forth as well. It was surprisingly common for quick ports and prototypes. Games like Karateka, Hat Trick, and Worms were written in Forth. Notably the Forth source code for Worms (1983) is available on GitHub.

The original Atari 800 Dig Dug was done in Forth. Professor Pac-Man (1983, Bally/Midway) was written in Forth. The Jupiter Ace, a home computer using FORTH instead of BASIC was released in the UK, and ports of many popular games including PAC MAN ran on the system. Bally and Midway, as well as Atari, often prototyped games in FORTH. The reason why some games are well-known to have been coded in assembly is because FORTH can define assembly routines as functions (words). In a 6502 Forth for example, you always write performance critical parts in assembler, as most Forth systems have an build in 6502 assembler (and if not, it's trivial to add).

RAMBrandt (Antic Software) was written in Forth.

AtariLab was written in Forth. Landon Dyer who worked at Atari said that someone at Atari would often start porting a game in Forth, but then later it would be optimized in assembly due to speed and size concerns of the 2600 and 800–The Atari 800 had a base of 16k RAM and the 800 had, I believe, 128 bytes of RAM– but the games game on cartridges containing 2k to 4k of ROM.

FORTH saw critical commercial success in the business world as well. RapidFile (1986, Ashton-Tate)

was a flat-file database manager written in Forth. Ashton-Tate was big in databases back then (dBase III fame), so this reached business users.

The open boot / open firmware on Sun SPARC stations and PowerPC Macs, as well as many embedded systems were written in FORTH. You dropped to an “ok” prompt on boot for diagnostics/config. Millions of machines ran Forth at startup without users knowing. *Millions.* Forth does have it's legacy to consider!

How was Star Forth written?

Here is the complete 8086 Assembly source code, commented, for fig-FORTH.

- fig-FORTH 1.0 source code from 1981

The forth interest group created various implementations, all listed here:

- other source code

This list includes implementations for: the 1802, 6502, 6800, 68000, 6809, 8088, 9900, the Alpha Micro, Apple II, Eclipse, IBM-PC, Nova, Pace, the PDP-11, and the VAX.

The point being proven, FORTH is out there with copious amounts of code in various architectures. It was thus easy to get a kind of FORTH running on the SD-8516. I quickly became obsessed with it. It took quite a while to iron out all the stack bugs. I'm still adding new things. We will see what tomorrow brings!