MOVING FORTH

Part 4: Assemble or Metacompile?

by Brad Rodriguez

This article first appeared in The Computer Journal #64 (November/December 1993).

"Keep it SHORT!" was the editorial directive for this installment. So I apologize for postponing the source listings to yet another issue. In the meantime, there is a new decision to contemplate:

How do you build a Forth system for the Very First Time?

You know now that most Forth code is high-level "threads," usually compiled as just a series of addresses. In the early days of fig-Forth, assemblers were often the only programming tools available. This was fine for writing Forth CODE words, but high-level threads had to be written as a series of DW directives. For example, the Forth word

```
: MAX ( n n - n)   OVER OVER < IF SWAP THEN DROP ;
```

would be written [TAL80]

```
      DW OVER,OVER,LESS,ZBRAN
      DW MAX2-$
      DW SWAP
MAX2: DW DROP,SEMIS
```

Later, as working Forth systems became widespread, Forthwrights began modifying the Forth compilers into cross-compilers [CAS80]. Thus with Forth on your CP/M machine (or Apple II, or whatever), you could write Forth programs for some other CPU...up to and including an entirely new Forth system for that CPU.

Because they create a new Forth from within Forth, these are often called metacompilers. Computer science purists object to this, so some Forthies use the terms "cross-compiler" and "recompiler." The difference is that a recompiler can only generate a new Forth for the same CPU.

Most PC Forths are now produced with metacompilers, but opinion is divided in the embedded systems arena [TIN91,ROD91,SER91]. The arguments for using assemblers to write Forth are:

1. Metacompilers are cryptic and hard to understand, and you must thoroughly understand a metacompiler in order to use it.
2. Assemblers are understood by the average programmer.
3. An assembler is almost always available for a new CPU.
4. Assemblers handle many optimizations (e.g. short vs. long branch).
5. Assemblers handle forward references and peculiar address modes; many metacompilers don't.
6. Assemblers use familiar editing and debugging tools.
7. The code generation is completely visible -- nothing is "hidden" from the programmer.
8. It's easier to tweak the Forth model, since many design decisions affect the internals of a metacompiler.

The arguments for metacompilers:

1. You write "normal" looking Forth code, which is easier to read and debug.
2. Once you understand your metacompiler, you can port it easily to new CPUs.
3. The only tool you need to acquire is a Forth for your computer.

The last is particularly applicable to those who don't own PCs, since most cross-assemblers require PCs or workstations these days.

I've written several Forths each way, so I'm painfully aware of the tradeoffs. I admit a preference for metacompilers: I find the Forth code for MAX much easier to read and understand than its assembler equivalent. Most of the arguments against metacompilers have been overcome by modern "professional" compilers, and if you're using Forth for work I strongly recommend investing in a commercial product. Alas, public-domain metacompilers (including my own) are still behind the times, clunky, and arcane.

So I'm going to take a radical position for a Forth programmer, and tell you to choose for yourself. I'll publish the 6809 code in metacompiler form, and I'll supply a metacompiler for F83 (IBM PC, CP/M, or Atari ST) [ROD92]. The Z80 code will be written for a CP/M assembler. The 8051 code will be written for a public- domain PC cross-assembler.

Forth in C?

No discussion of this topic would be complete without mentioning a new trend: Forths written in C. These have the advantage of being more portable than assembler -- in theory, all you have to do is recompile the same source code for any CPU. The disadvantages:

1. Less flexibility in the design decisions; e.g., direct-threaded code is probably not possible, and you can't optimize register assignments.
2. You have to recompile the C source to add new primitives.
3. Forth words carry the C call-and-return overhead.
4. Some C Forths use inefficient threading techniques, e.g. a CASE statement.
5. Most C compilers produce less efficient code than a good assembly-language programmer.

But for Unix systems and RISC workstations, which frown upon assembler, this may be the only way to get a Forth up and running. The most complete and widely used of the public-domain C Forths *[at the time of publication]* is TILE (TILE_21.ZIP, file #2263 on GEnie's Forth Roundtable). If you're not running Unix, you should look instead at the Genie files HENCE4TH_1.2.A (#2490) and CFORTHU.ARC (#2079).

To continue the previous comparison, here's the definition of MAX from HENCE4TH [MIS90]. I omit the dictionary headers for clarity:

```
_max() {
    OVER  OVER  LESS IF  SWAP  ENDIF  DROP }
```

Instead of assembler, C is used to write the CODE words in the kernel. For example, here is HENCE4TH's SWAP:

```
_swap() {
    register cell i = *(dsp);
    *(dsp) = *(dsp + 1);
    *(dsp + 1) = i;
}
```

(Please note: there is quite a variety of techniques for writing Forth words in C, so these words may appear radically different in CFORTH or TILE.)

On a 68000 or SPARC, this might produce quite good code. On a Z80 or 8051, quite the opposite. But even if you plan to write a Forth in C, you need to understand how Forth works in assembler. So stay tuned for the next installment of Moving Forth!

REFERENCES

[CAS80] Cassady, John J., METAFORTH: A Metacompiler for Fig- Forth, Forth Interest Group (1980).

[MIS90] HenceFORTH in C, Version 1.2, distributed by The Missing Link, 975 East Ave. Suite 112, Chico, CA 95926, USA (1990). This is a shareware product available from the GEnie Forth Roundtable.

[ROD91] Rodriguez, B.J., letter to the editor, Forth Dimensions XIII:3 (Sep/Oct 1991), p.5.

[ROD92] Rodriguez, B.J., "Principles of Metacompilation," Forth Dimensions XIV:3 (Sep/Oct 1992), XIV:4 (Nov/Dec 1992), and XIV:5 (Jan/Feb 1993). Note that the published code is for a fig-Forth variant and not F83. The F83 version is on GEnie as CHROMIUM.ZIP

[SER91] Sergeant, Frank, "Metacompilation Made Easy," Forth Dimensions XII:6 (Mar/Apr 1991).

[TAL80] Talbot, R.J., fig-Forth for 6809, Forth Interest Group, P.O. Box 2154, Oakland, CA 94621 (1980).

[TIN91] Ting, C.H., "How Metacompilation Stops the Growth Rate of Forth Programmers," Forth Dimensions XIII:1 (May/Jun 1991), p.17.

Author's note for web publication: the files formerly available on the GEnie online service are now available from the Forth Interest Group FTP server, ftp://ftp.forth.org/pub/Forth. Also, several new Forths-in-C have been published since this article was first written. Consult the "systems" FAQ at ftp://ftp.forth.org/pub/Forth/FAQ for a current list.

Continue with Part 5 | Back to publications page