Forth Code Size#

from A.N.A.L.O.G. 11/83

My nerdy little brother says that a program written in FORTH can take up less memory than the same program written in machine code! I bet him a week's allowance that he's wrong Do I win?

Nervous in Nevada Looks like you'll have to split the kitty, kids. it is possible for a FORTH program to take up less space than machine code - but it isn't very likely on an ATARI computer. Let's take a look at the factors that control the code efficiency of FORTH and M/L programs.

At the innermost core of every FORTH system is a block of machine-language routines called the kernel. The kernel can be thought of as FORTH's operating system; it maintains the stacks, controls memory allocation and performs all the dirty little housekeeping duties that make FORTH look like FORTH. When you compile new words into a FORTH dictionary, all you are doing is defining new execution patterns for the fundamental FORTH routines inside the kernel.

Each of the FORTH systems available for the ATARI come with a "bare bones" kernel of fixed size. For example, the valFORTH 1.1 kernel takes up about 7.5K of RAM; the Team ATARI figFORTH kernel, about 8.9K. Because the kernel must be permanently linked with your program in order for it to run, the size of the kernel determines the absolute minimum size of your program. So even if you wrote a valFORTH program consisting of just one word:

```
( Change background color to black )
: PROGRAM 0 710 C! ;
```

the final product would still occupy at least 7.5K! Machine language laughs at the idea of a kernel. It requires no overhead at all because it communicates directly to the 6502 microprocessor at the lowest possible level. In fact, you can write a little routine in assembly like this one:

```
A900     LDA #0
8DC602   STA 710
```

that performs exactly the same function as our FORTH word PROGRAM in only five bytes.

It's the size that counts.

When you compare the sparseness of machine code to FORTH's kernel overhead, it's hard to conceive of an application where FORTH would be more efficient. Yet even as the kernel taketh away, the kernel giveth - in the form of generality.

A FORTH kernel is like a Swiss Army knife. It consists of a number of all-purpose tools (subroutines) built around a simple, versatile control structure (the stack). A knowledgeable FORTH programmer can exploit the built-in features of the kernel to concisely implement all sorts of elaborate procedures, much as an assembly hacker uses OS subroutines as often as possible to simplify his work. But a FORTH kernel is far more versatile than a machine-specific OS; and although the initial size of a FORTH program is large, its threaded structure makes it grow less quickly than a machine-language program. Theoretically, a point can be reached where FORTH and machine code take up the same amount of RAM (see Figure 1), after which the FORTH application will require less memory than the same job written in pure machine code.

When is this break-even point reached? It depends on the nature of the application, the power of the FORTH kernel and the skill of the programmer. Generally speaking, your FORTH program has to get fairly large before it will begin to compete with the best RAM-cramming efforts of an assembly hacker. That's why you're not likely to realize the potential efficiency of FORTH with a computer as small as the ATARI. On larger machines, however, FORTH can make a big difference when it comes to saving memory.

One way to reduce the size of a FORTH program is to run it through a utility called a target compiler. A target compiler analyzes your application and strips away all the unnecessary gunk and dribble, leaving you with a tight package of object code that is smaller and maybe even a little faster than the original. None of the FORTH systems available for the ATARI offer a target compiler, although Valpar International is supposed to have a valFORTH compiler in the works.

As I mentioned above, the threaded architecture of FORTH makes a big contribution to its efficiency. But it takes good, structured programming techniques to realize this benefit. We'll discuss the controversial subject of structured programming in my next installment.