Forth in a Hardware-Rich Environment

Mitch Bradley

Sun Microsystems
2550 Garcia Avenue
Mountain View, CA. 94043

Historically, Forth has excelled on computers with limited hardware resources – small memories/16-bit address spaces, small or nonexistent mass storage devices and CPUs capable of executing less than 1 million instructions per second (1 MIP). Forth has done well in this kind of hardware environment and will probably continue to be a language of choice for such small machines.

However, computer hardware is getting cheaper and more powerful each year. Already a 64K memory is considered small. Microprocessors are getting faster and faster, and their memory address space is well into the several megabyte range. During 1985, systems based on the Motorola 68020 or the National 32032 will boast a full 32-bit address space (4 gigabytes!) and CPU speeds of 2 MIPs or more. Mass storage devices are similarly improving. It is already possible to buy a hard-disk drive that stores over 100 MBytes and fits in the space of a standard-height floppy drive.

Forth, as most of us know it, doesn't know what to do with a machine like this. If Forth is to have a future in applications other than machine control or real-time systems, it must evolve to take advantage of the hardware capabilities of machines like these. For the purposes of this review, such a hardware-rich machine will be called a "workstation".

Currently there are several aspects of Forth that do not match well with workstations. Many of these problems are not inherent in the Forth language itself, but rather parts of the Forth "culture", which is accustomed to having to live with very limited hardware.

Problem No. 1: The 16-bit Syndrome

A 16-bit address space is too small for even the current generation of microprocessors (20 to 24 bit address busses), and it is ludicrous for the next generation of 32-bit microprocessors. For a 32-bit memory interface using 256 Kbit dynamic RAMs, the smallest memory possible is 1 Mbyte! True, many interesting things can be done with only 64 K of memory, but there are also lots of applications whose data won't even begin to fit in 64 K.

Unfortunately, most Forth systems, including the 83 Standard, explicitly assume or specify that the stack width is 16 bits. Because this assumption is buried so deeply, it is not possible to produce a "standard" Forth system with addresses or stacks wider than 16 bits.

Problem No. 2: No Files

Remembering block numbers on a 50 MByte disk is next to impossible. Restricting source code to fit into a 16 line by 64 character format makes no sense on a computer with fast disks. The 1K block notation is an artifact of the days when microcomputers all had low-performance I/O systems.

True, blocks are still useful in some applications like data acquisition, but for many other applications, they are extremely cumbersome. Named files of arbitrary length are the obvious solution to this problem, but there is no standard for how to deal with files in Forth.

Problem No. 3: The "Do Everything Yourself in Forth" Mentality

Most machines today come packaged with a lot of software. Some of it is very good. The Forth community does not have a strong notion of coexisting with and complementing existing software systems. Instead, the idea prevails that other languages/operating systems are somehow "bad" and that every bit of software should be written or rewritten in Forth. Theoretically, this might be nice; it isn't going to happen. There is just too much software out there to reinvent it all in Forth.

This attitude is hurting Forth because it tends to isolate the Forth community from the mainstream of what is happening in the software world. Besides the attitude problem, there is an associated technical problem. Since the only mass storage access that the Forth standard specifies is blocks, and since no other applications software that I have seen uses blocks, there is no standard way in Forth to communicate with the rest of the software world.

Problem No. 4: The "Clever Bit-Hacking" Syndrome

It has become standard practice to encourage Forth programmers to be "clever" and to save bytes at every opportunity. The trouble with this is that many byte-saving techniques hinder portability by taking advantage of implementation details of the particular Forth system. This in turn makes it necessary to modify the program to make it run on a different Forth system. Forth programmers have not tended to build on top of each other's work to a great extent, perhaps because of the portability problems introduced by being too clever.

A particularly prevalent example of "cleverness" is that of using an existing Forth word to do something for which it was never intended. For instance, it is possible in many Forths to use the word COUNT to step through an array of bytes. This takes advantage of the implementation detail that the length field of a packed string is the first byte in the string, and the address of the actual characters in the string is the next byte address. This trick does not work if a Forth implementor has chosen a different representation for packed strings. In addition, such usages can be confusing to read because they require intimate knowledge of implementation details.

Sometimes such techniques are necessary to make a program fit on a particular machine, but too often they are done when they are not necessary.

Problem No. 5: No Electronic "Standard Interchange Format"

We need to be able to share code with each other. Publishing Forth screens is better than nothing, but typing them in to your machine is slow and error prone. It just doesn't work for anything longer than a few screens. The CP/M world has Modem 7, the Unix world has UUCP mail, the Forth world has typing. If Forth programmers are ever going to routinely share significant (i.e., long) programs, some better interchange medium has to be used.

Solutions:

Forth has a lot to offer to the world of powerful computers. I would very much like to see Forth become a popular language on workstations. I therefore offer the following solutions to the problems just mentioned.

Solution A: Remove the 16-bit Restriction

Remove the 16 bit restriction from the standard. It is possible to describe Forth stack operations as operating on "stack items" instead of on "16 bit numbers". This, in conjunction with a few extra words and a slight shift in the way we think about addresses and data, can result in a Forth standard that is equally applicable to 16 or 32 bit machines.

Using techniques described in [1,7], the author and others have written code which runs unmodified on both 16 bit and 32 bit machines. It is not hard to do, but it requires a clear understanding of the difference between the size of a stack item and the size of a datum stored in main memory.

Solution B: Standardize a File System Interface

Standardize a portable file system interface. There are two distinct objectives that a file system interface needs to satisfy. One is to allow Forth programs to access files in the same way regardless of which Forth implementation is being used. The other objective is to allow a Forth program to access and manipulate files used by other utilities which may be available on a particular machine. These objectives are roughly equal in importance and they need not be mutually exclusive.

One popular notion is to have a Forth file system built entirely on top of the BLOCK system. This approach fails to satisfy the second goal. Another approach is to duplicate the file system access calls of the particular operating system your favorite machine runs. This solves the second problem, but doesn't solve the first.

The key to the simultaneous solution of both goals lies in choosing what to standardize and what to leave unspecified. The file system has to specify a model for a file and a set of standard words for accessing such files. The model has to be simple enough to be compatible with files from nearly every existing operating system, and powerful enough to be useful for the majority of applications. For standalone applications, the file model would be implemented on top of BLOCK, but Forth systems that run under some other operating system should implement the file model using the files already provided by that operating system.

Here is a good model: A file is an array of bytes numbered from 0 up to the number of bytes in the file. No other sub-structure is imposed by the file model; for example, there is no notion of a "record". Records and such can easily be built on top of the basic "array-of-bytes" model needed.

The basic operations on such a file are opening, closing, reading or writing a specfied number of bytes and setting the position within the file for subsequent read or write operations. For a further elaboration of such a file system interface, see [2,3]. The referenced file system interface may be implemented on top of just about any popular operating system and the Forth programs which use the interface will be portable from machine to machine. The Forth programs can access files created by other utilities, and vice versa.

Solution C: Attitude Adjustment Hour

The Forth community must realize that Forth is not going to take the software world by storm and supplant all other systems (Unix, CP/M, MS/DOS, C, PASCAL, etc.). The best we can hope for is that Forth will become widely accepted as a tool on equal footing with these other, more established tools.

This means that we, as Forth enthusiasts, must abandon our cherished hope of solving the whole universe of software problems with Forth and get on with the task of making Forth work with these other systems. We must rid ourselves of the notion that other languages and operating systems are an evil conspiracy shoved down the throats of an unsuspecting public by the dark side of the force.

Once we have a spirit of cooperation, we can start to freely use software other people have written in other languages. This includes editors, print spoolers, spreadsheets, text formatters, ad infinitum. This does not preclude us from going ahead and implementing such tools in Forth, but we must not fool ourselves into thinking that something is better just because it's written in Forth. "Reinvention of the Wheel" is only justified if the wheel can be improved in some way.

The main technical problem involved is sharing data with other programs. This is solved by a suitable file system interface as described earlier. Please note: I do not advocate simply constructing Forth words which mimic the file system access calls of your favortie machine. I propose a file

system interface which is the same on any Forth system on any machine, but which uses and accesses the files already provided by the "native" operating system on the machine. See [4] for proof that this is possible.

Solution D: Restrict Cleverness

Don't be clever unless you really need to be. Use Forth words to operate only on the data structures they were intended to access; don't take advantage of things that "just happen to work on my machine". Invent new words that do what you need, even when they happen to be the same as some other existing word. For example, inventing a word NEXT-LINK which just does a fetch (@) is clearer and easier to modify later than just using @ directly.

Doing things explicitly makes it much easier to move programs from machine to machine, and thus makes it more likely that people will build on each other's work.

Solution E: Standard Interchange Format

The only hope that I see for such an interchange medium is the phone line. We need a standard protocol for transferring data over the phone line. I see no reason not to use the Christensen (Modem 7) protocols [5] already established in the CP/M world. The only problem is that the Christensen protocol assumes the transfer of files, not blocks, which brings us right back to the need for a file system.

For a lot of people, the file system interface alone solves the interchange problem. Many operating systems already have a file transfer program which can talk to many other machines and which transfers native operating system files. If Forth programs can access those native fields, the problem is solved.

Summary

Forth needs to break out of the tiny machine mold if it is going to be used on the next generation of machines. The major things that Forth needs in order to move to the next generation are:

1) the removal of the 16 bit orientation from the standard,
2) a suitable file system,
3) a cooperative attitude toward other languages,
4) less use of implementation-dependent knowledge,
5) an electronic means of sharing programs with each other.

References

1. Bradley, Mitch, and Sebok, Bill. "Extended Addressing Wordset," *Proc. 1984 Rochester Forth Conference*, 1984, p. 284.
2. _____. "A Portable File System Interface for Forth," *Proc. 1983 Asilomar FORML Conference*, 1984, p. 231.
3. _____. "Operating Systems Working Group Report", *Proc. 1984 Rochester Forth Conference*, 1984, p. 291.
4. Kernighan, Brian, and Plaugher, Bill. *Software Tools*. Prentice-Hall.
5. Christensen, Ward. "Modem Protocol Overview". Unpublished, but available on various CP/M electronic bulletin board systems.
6. Taylor, Robert. "Send and Rcv: A Forth Implementaion of the XMODEM Protocol," *Dr. Dobb's Journal*, Vol. 8 No. 9, Sept. 1983, p. 82.
7. Bradley, Mitch, and Sebok, Bill. "Compatible Forth on a 32-bit Machine," *Journal of Forth Application and Research*, Vol. 2 No. 4, 1984.

Manuscript received July 1984.