Introduction

The focus of this issue is the "Forth engine." As Forth has been a software implementation of an extendable instruction set hardware architecture, there are a number of conceivable ways to implement such a machine. As an introduction to these papers I'd like to present a discussion which might serve as a groundwork for the authors and, in so doing, attempt to place in perspective some of the possible approaches. I'll pose the following questions:

1. What is a Forth engine?
2. What are the potential advantages of one?
3. What are the reasonable avenues to one?
4. What are the areas critical to achieving the best results?

What is a "Forth Engine"?

The term "Forth engine" will take on many meanings in these articles. I'll use the term to mean "a board set or chip which will allow a board or board set to execute Forth as its primary programming language." Using this definition there are six primary approaches that could be taken in the building of such an engine. Each has its strengths and weaknesses.

What are the Potential Advantages of a Forth Engine?

Many different goals can be sought in the design of a system. Among them are speed, size, cost, and usability. The Forth language tends to answer the usability problem, else there would not be much interest in doing a machine specifically for the language. The speed, size, and cost aspects, this is after all the real world, tend to fit the "pick any two" rule.

Higher speeds should be attainable when one commits a software emulator to hardware. This has certainly been proven by the bit slice machines presented in the following papers. To some degree, the bit slice machines also answer the size problem. It would have taken a much larger machine to replace these systems.

At the other end of the spectrum, the single chip implementations can answer the size and cost combination. If maximum performance is not a goal, they provide an elegant solution to many problems.

Finally, the size and speed combination might eventually be solved by a CMOS implementation of the Forth architecture. In fact, if one assumes that funding can be found for the development, the end user might find that it is possible to achieve a solution which addresses the three part problem.

What are the Reasonable Avenues to a Forth Engine?

ROM'd Forth for a "Standard Processor"

The first approach is to provide an existing computer with an ROM'd Forth system. Forth becomes active at the time the machine is booted. The type of Forth implemented

for such a machine may be the simple FIG model which tends to be transportable and small, but single-user and slow. Many applications are well served by such a system. This approach also allows the placing of a very complete and highly optimized multitasking/ multi-user Forth into an existing machine, thus allowing the solution of a very wide range of problems.The cost and throughput of such machines can vary widely. Processors used have varied from the Z-80 to the 68000. Sieve of Eratosthanes times range from greater than 200 seconds to 9.6 seconds for 10 passes. Machine cost to the end user ranges from roughly $600 to $2000 or so.

The development cost inherent in this approach lies primarily in software and will be much higher for the optimized non-FIG model system. This approach is, however, an economically reasonable one for the system developer who must use existing hardware, who is unable to bear the cost of a processor hardware development, or who needs a more versatile machine than is available in one based around a single chip computer.

The Single Chip

The single chip computer was designed primarily for inclusion into systems as peripheral interfaces, sequencers, or controllers. They provide a cost effective means of replacing single purposed, hard wired logic. The use of such devices has enormous impact on the economics and flexibility of many products.

The traditional means of programming single chip computers includes assembly and targetted high level languages or cross-compilers. The steps involved in cross- or target-compilation can result in extreme confusion during the debugging phase of product development. In-circuit emulators (ICE units) are a helpful if expensive aid during this time of trial.

The concept of an interactive high level language on board the single chip machine is a very attractive one because it can simplify the product development process. At least two Forth engine developments have occurred in this area, the Rockwell R65F11 and the Micromint Z-8, with another in progress, the F68K.

The cost of devising the code and providing it in a masked programmed part is not small, but when spread across a potentially large number of users, the part's cost can be quite reasonable. Thus, the system developer has a tool that decreases his development cost while not markedly increasing the end user's cost; a nearly ideal situation.

The kind of Forth that it is possible to place inside a single chip processor is severely restricted by the size of the ROM on board the processor, usually 2 to 4K bytes. As a result, such systems have been implemented along the lines of the FIG model because of its very small kernel size. Thus, while the devices are small, and relatively inexpensive, they tend to represent the low end of the performance spectrum; just what is needed in many controller situations.

The Microcoded Standard Engine

Microcode is the most fundamental software structure in a computer. It consists of bit patterns which control the flow of data through the computer's busses, buffers, ALU and registers. The most common form of microcode has two dimensions. The machine instruction being executed provides an entry point into a list of bit patterns, or microinstructions, which must be used in the execution of the steps required to do the machine instruction. This is called vertical microcode. One machine language instruction may require the execution of a number of microinstructions. The execution of each microinstruction usually takes one clock cycle for the machine, often referred to as a "microcycle."The bit pattern found for each step actually controls the hardware to allow a desired data flow within the computer. This is called horizontal microcode. Microcode is fundamentally linked to the structure of the computer. The horizontal microcode must contain as many bits as is required to control all

of the data flow that is needed by the architecture of the machine. Microinstruction lengths of 60 to 80 bits are not unusual.

The need for increased performance in some applications has driven the search for ways to implement faster Forth systems. From the programmer's point of view, the most effective way to get maximum throughput from a machine is to have access to the microcode that forms the basic instruction set of the computer. This allows optimization of those groups of assembly language instructions which are critical to the performance of the system. These may then be executed as efficiently as is possible given the architecture of the processor.

It is unfortunate that few computers allow the programmer access to the microcode control store, the place where the machine's instruction set is defined in terms of microcode. Some machines are constructed with a feature called "writable control store" (WCS). However, even when a machine has a WCS, it is not uncommon for the vendor of the machine to reserve its use by failing to provide documentation for it, or by not providing support for those installations which have their own microcoded routines. Indeed, the customer use of a WCS can play the devil with the vendor's service organization if the customer makes the computer into a different machine than provided by the vendor. This is certainly possible with writable control store.

Thus, such an approach can be taken if the machine allows it, the vendor will support it, and if you know something about microcode. This may be a tough group of skills and circumstances to put together. The technical note on the development of a Forth for the NCR/32, which has a form of writable control store, discusses the results of such an effort — a machine with enormous throughput for many simultaneous users.

The Bit Slice Machine

Bit slice technology has allowed the development of custom architectures for minicomputer level products beginning with a few relatively standard parts. For example, the PDP-11/34 of Digital Equipment Corporation is fabricated from bit slice elements.

A "bit slice" is an integrated circuit that contains a register element, data busses, and an ALU element. Each has the width in bits of the slice. More than one bit slice can be concatenated with some extra logic into a processor of "n" bits in width, where "n" is commonly 16, 24, or 32.

Bit slice machines are microcoded by the designer to allow the specific machine language instruction set sought by the design. Microcycle times of 80 to 125 ns are now common for bipolar bit slice machines when the control store is located in ROM having 55 ns or faster access time. Thus the designer has already complete freedom to develop the architecture and language support facilities within the machine and still know that the end product will likely be very fast!

While the throughput achievable with this technique is very high indeed, this approach is not for the faint of heart, the lean of pocketbook, or the short of experience. The end result, too, is not inexpensive. However, it may well be justifiable for some applications given the performance benefits. Such machines are appropriate where no off-the-shelf technology can solve the existing problem.

The Discrete Component Machine

There are fundamental restrictions to the speed attainable if one uses a bit slice approach in the fabrication of a computer. The first is the logic family that can be used. Typical bit slice building blocks use bipolar TTL logic. They currently have a limitation in minimum microcycle time which is largely due to that logic family. In order to employ a faster logic family, such as ECL, it is practically necessary to do a discrete logic development, that is, to use small and medium scale logic instead of the bit slice building blocks.

While discrete logic design is not as convenient as a bit slice application for the designer, a further level of architectural optimization can occur. This optimization for function can take place at two levels. Internal simplification, read optimization, is the first level. A bit slice design really does expect to be completed with a vertical and horizontal microcode scheme. If, for example, a discrete component design is done with appropriate planning, some of what would have been horizontal microcode bits can be part of the machine language instruction word.

The second level is that of special hardware which can be applied to the design which is not part of the bit slice device. A barrel shifter or hardware memory limit protection hardware would be examples of such additions. Such additional hardware can either provide helpful function or speed through the parallelism of certain operations. The disadvantages of such an approach are that they tend to be rather expensive, and because of the logic families used to gain speed, they tend to consume huge amounts of power. If one needs a special engine to do supercomputer level crunches for special (read "very well funded") applications, this may be a viable approach.

The Custom Processor Chip

As the problems with bit slice and discrete logic implementations become understood and solved, the final step in the evolution toward a special Forth engine on a chip looks more possible. The advantages of a custom CMOS processor would be legion. This is particularly true as it becomes practical to use 1.6–3 micron design rules. Supercomputer throughputs seem possible.

Needless to say, the endeavor will be enormously expensive. It's hardly possible to accomplish such a project without extensive support. Perhaps the success achieved with the bit slice machines will provide the documentation required to inspire the well-heeled to take an interest.

What Areas are Critical to Achieving the Best Results?

What are we going to define as "best"? Again we must analyze the goals of the individual approaches. Are we trying to improve a product development environment, make the speediest machine, or are we trying to provide the language with good correspondence to the standard while being pretty darn fast? It seems to me that the trade-offs are pretty obvious for the ROM'd versions of Forth for standard or single chip standard processors. There will be strong opinion. Fortunately, there are also enough choices to satisfy most people.

The bit slice machines provide us with some insight into the problems that must be addressed in the long run. Here are some things to look for:

1. What compromises were made in what we think of as Forth to ease implementation. An example would be byte versus word structuring of the machines. These might raise some interesting philosophical questions about which driving force is most appropriate for the continued development of the standard.
2. Brute speed is not the only requirement for utility in the real world. The ability to communicate with the world outside the computer is of great concern. How do these developments handle I/O and interrupts?
3. Have these developments addressed the problems of multi-user or multitasking environments? What do they tell us about device/user binding?
4. Do they solve a real problem in a way that makes the approach attractive for other applications?

Finally — Words of Thanks

It has been a great pleasure for me to have been involved with the authors whose work is presented in this issue. I sincerely appreciate everyone's patience through the last months. The authors have been very cooperative. The *Journal* staff has simply been super. My thanks to everyone!

Michael K. Starling