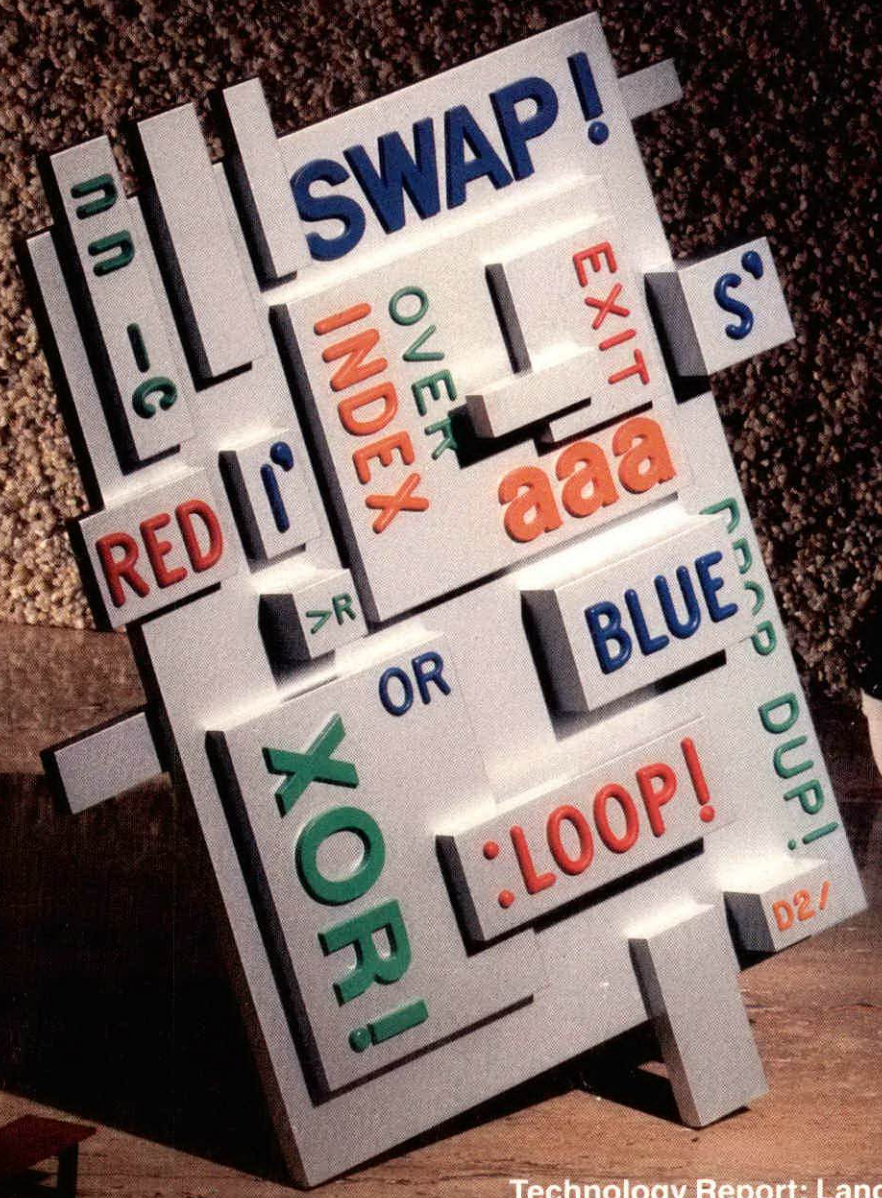
A HAYDEN PUBLICATION

ElectronicDesign®

FOR ENGINEERS AND ENGINEERING MANAGERS — WORLDWIDE

MARCH 21, 1985

Forth language shapes the structure of 10-MOPs chip



Technology Report: Languages for artificial intelligence seek to become native machine tongues

Product Report: Single-board computers burst with new functions

Autozero chip slashes offset voltages of stand-alone op amps

BEHIND THE COVER

I always assumed that someone would build a computer to run Forth," observes Charles Moore, who invented the high-level language 15 years ago. "In fact, several microprocessors already emulate the Forth architecture." But in the end it was Moore himself who actually sat down with fellow enthusiasts and determined that a Forth processor was feasible.

The Novix NC4000A, what that company's John Golden calls "the first computer for which the software was written first," is the subject of this issue's cover story (p. 127). It uses an instruction set consisting of subroutines (or "words") taken directly from Forth. The user's program feeds those subroutines via a compiler into an instruction latch that actually controls the machine. All time-consuming detours through lookup tables and microcode are eliminated.

When Moore 's team got going four years ago, they spent one week outlining an architecture that would exploit the highly efficient data and address stacks for which Forth is famous. Another year passed before they could decide which of two dozen variations was best.

"It was hard to get financing because of the novelty of the design," recalls Golden. Nonetheless, work went ahead. Moore talks of a half-dozen iterations, "since each design turned out to have a better and more interesting one hiding behind it."

The pace picked up when Sysorex International stepped in with funding and Bob Murphy of Torric Corp. was brought in to help with the hardware aspects. From late last March through July, he and Moore virtually sequestered themselves from the world. By that time, Moore had hit upon a parallel architecture that Murphy dubs "octopuslike, because it can move in many directions at once." A major contribution of Murphy's was the elegant programmable I/O configuration.

In August they released the layout to Mostek, and packages became available just before Christmas. "It worked the first time as predicted," comments Murphy. "It was intended as proof in principle," adds Moore, "but it turned out to be a very nice little computer." In fact, Moore finds the chip "surprisingly comfortable to work with" and "blindingly fast." But, conceding his own likely partiality, he looks forward to "getting more feedback" on its highly unusual features.

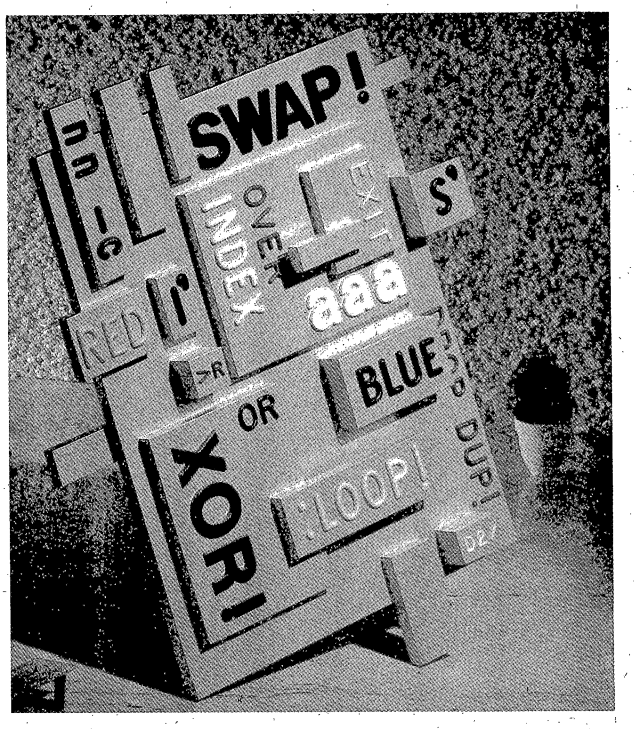
Fast processor chip takes its instructions directly from Forth

Executing 10 million operations/s, a chip compiles Forth subroutines into op codes, whose bits stir simultaneous activity in the ALU and memories.

Programs written in high-level languages generally have to be translated into the instructions of the computer they run on. But the relationship between the software commands and the machine instructions is far from perfect: A single command may evoke a sequence of instructions.

For better performance, an engine is occasionally designed to run portions of a particular high-level language actually in microcode. But now, for the first time, a high-level language, Forth, has been put directly, without even microcode, into the logic of a single-chip processor. In other words, a Forth program becomes the chip's external microcode.

Moreover, while the Forth language is itself optimized for speed, the 16-bit Forth processor goes one big step beyond. Its parallel, bit-slice-like architecture lets it run 100 times faster than emulated Forth processors and also outperform conventional devices. An initial CMOS gate-array version runs a benchmark al-



John Golden, Novix Inc.
Charles H. Moore, Consultant
Leo Brodie, Consultant

John Golden is general manager of Novix, a Cupertino, Calif., partnership formed to develop and market the NC4000 family of processors. After earning a BSME from the University of Michigan, he spent three years with NASA and 17 years with Systron Donner's aerospace group.

Charles H. Moore graduated with a BS in physics from MIT and did graduate work in mathematics at Stanford. As an independent programmer in the early 1970s, he created the Forth language and cofounded Forth Inc. Lately his energy has been spent on the NC4000 high-speed Forth engine.

Leo Brodie, a former technical writer at Forth Inc., has written two books on Forth programming, Starting Forth *and* Thinking Forth *(Prentice-Hall). He now is a consultant in Forth programming.*

Reprinted from **ELECTRONIC DESIGN** - March 21, 1985

Cover: Forth processor chip

gorithm, the Sieve of Eratosthenes, written in Forth in 0.3 s, while the 68000 runs it in assembly language in 0.49 s.

Thus in one bound the NC4000 family of high-speed processing engines leaps two barriers. First, it rids the system designer of the need to build and program a custom processor, permitting the use of an off-the-shelf microprocessor with comparable performance. Second, by executing a widely used high-level language directly, it frees the programmer from the difficulties of writing in assembly language.

The first chip to be released, the NC4000A, runs at a clock speed of 8 MHz. Each instruction executes in a single clock cycle, 125 ns, and performs as many as five operations simultaneously—for a chip speed of over 10 million operations a second. Subsequent versions should improve that rate by an order of magnitude.

Such blazing speeds will make the Forth processor suitable for applications that might otherwise be designed in bipolar MSI or bit-slice chips. The CPU of a supercomputer is a case in point. Here the chip's programmability adds the advantages of adaptability to ever-changing design specifications, quick turnaround, and reusable code for later-generation products. These advantages also pertain to CAE gear and to artificial intelligence applications. Finally, the chip's programmable I/O buses allow an array of the processors to be tied together in parallel, promising superfast computing.

Forth source code consists of numbers and words. In Forth terminology, a "word" applies equally to operators like "add" (written +) and to subroutines (see "Words, words, words," below). The Forth processor executes many Forth words with one instruction in one clock cycle, and it also executes the calls to subroutines in one clock cycle.

The execution rate is due to three of the chip's architectural features. First, five parallel buses

Words, words, words

Forth is a high-level language widely used in real-time control and other high-performance applications. Its most distinctive features are a dictionary of Forth words, which the programmer expands at will, and a data stack for passing arguments between those words.

Forth source code thus consists of words and numbers, separated by space. In the phrase 2 2 + the symbol + is dubbed a word with the meaning "add," and its two arguments precede it. Forth words assume that their arguments will be waiting for them in a strictly last-in, first-out order on the data stack. Accordingly, the Forth compiler, upon encountering 2 2 , pushes the numbers into the top two positions on the stack, from which the word + pops them, ultimately storing the solution (4) in the top stack position.

A special group of defining words lets the programmer add to the built-in dictionary. The most powerful of these is : , which is used to define new words in terms of earlier ones. A new word named LIFT might be defined thus:

```
: LIFT HAND OPEN ARM LOWER
  HAND CLOSE ARM RAISE;
```

The semicolon marks the end of the definition. The Forth compiler converts this code into a list of the dictionary addresses of the component words, producing threaded code. The new word, LIFT, now joins the linked list of its predecessors and may be used instead of the long sequence of words that constitute its definition.

Forth words can be nested like this almost indefinitely. Writing a Forth program consists of building increasingly powerful definitions, such as this one, thereby creating a dictionary unique to the application. As a bonus during development, a programmer can test each word (no matter how low-level) by placing its input arguments on the stack and invoking the word. Afterward, the results appear on the stack again and are easily displayed.

On rare occasions, the data stack does not suffice. So another defining word, VARIABLE , is available to create storage locations for variable data. For example, once VARIABLE ORANGES has been defined, the programmer would use ! (pronounced "store") to store a value in the location, thus 50 ORANGES! . Fetching the contents of this location is the job of the word @ (pronounced "fetch"). In this case ORANGES @ would fetch 50 as the value of the oranges.

Forth provides many other data structure operators. But more importantly, it provides the tools necessary for the programmer to create whatever data structures the application needs.

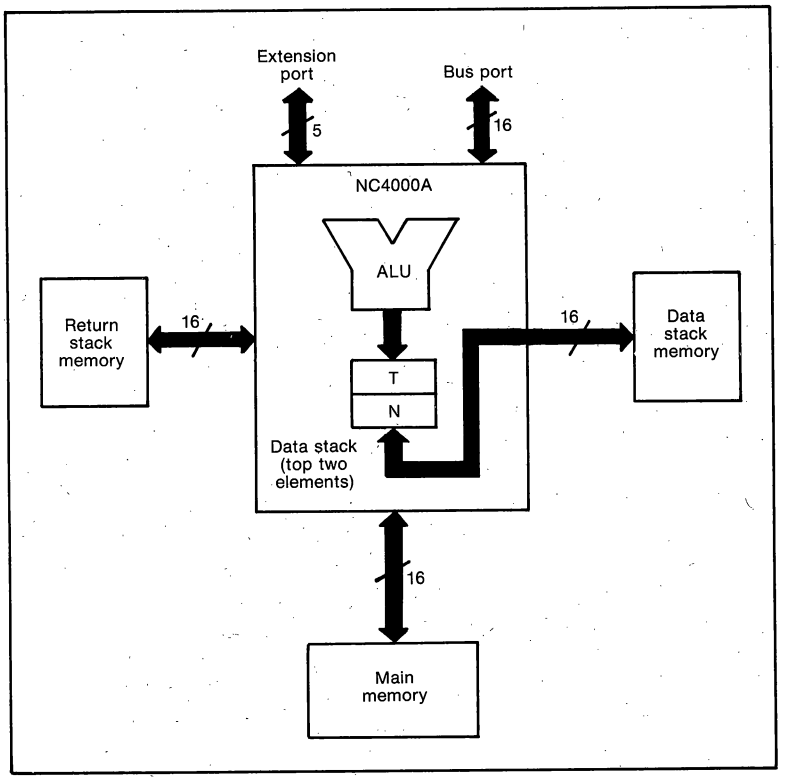
The language also includes words essential for structured programming. Among its control structures are jumps, such as flag IF aaa THEN , counted loops such as limit index DO aaa LOOP , and indefinite loops, such as BEGIN aaa flag UNTIL. In addition, since Forth is extensible, the programmer may easily define application-dependent control structures such as multiple-exit loops and case statements.

give each instruction simultaneous access to main memory, a data stack, a return address stack, and two I/O buses. Second, as in a bit-slice approach, an instruction register exercises direct control over the ALU and memory addressing. Third, an instruction sequencer—actually a hardware implementation of a routine that exists in software in previous Forth systems—hastens the processor's execution of subroutine calls.

In contrast, when Forth is run on a conventional microprocessor, its words are distinct from the processor's instruction set. They not only have to be compiled into those instructions (often less than optimally) but also may require a series of instructions taking many clock cycles. Moreover, although the Forth implementation would allot areas in main memory to a data stack and a return stack, it would have only one bus over which to gain access to all three. Finally, issuing a machine-level subroutine call or returning from one can take a conventional microprocessor as many as 15 clock cycles to execute.

In what follows, "subroutine" virtually always is used synonymously with a Forth word; "definition" refers to a Forth word defined in terms of previously existing words; an "instruction" refers to a word's machine-language correlate—one or more 16-bit op codes—on the Forth processor; and an "operation" refers to any one of the 40 activities that individual bits of the op code can control. Combinations of up to five of these operations create a set of over 130 single-op code instructions—more, if the permutations of register addressing are included.

Also, the terms "compiler" and "interpreter" have a slightly unusual sense. A compiler refers to the software routine used to transform Forth source code into threaded Forth object code for the NC4000; in this object code, each definition becomes a list of op codes, some of which are subroutine calls. The interpreter is a routine



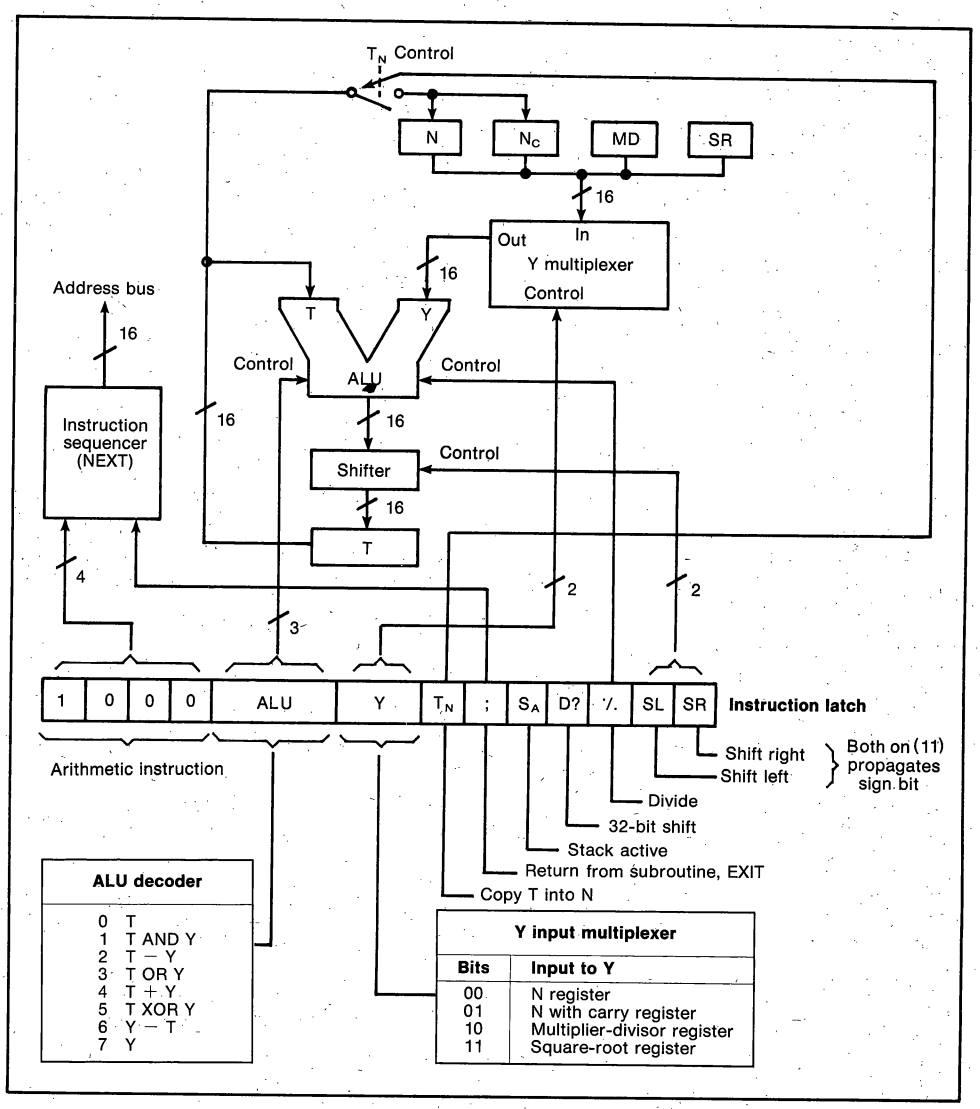
1. The NC4000 Forth processor uses separate buses to main memory, its data stack, and its return stack, as well as to two I/O ports. That parallel architecture lets instructions execute multiple operations simultaneously.

Cover: Forth processor chip

that looks up the address of each word entered from the keyboard (or read from mass storage) and executed in turn. Thus Forth runs at the speed of compiled languages while allowing the programmer to invoke definitions and create new ones at any time from the keyboard.

The parallel bus architecture of the Forth processor builds on the language's highly structured (and streamlined) approach to handling data and addresses. Like reverse Polish notation, it always places operands ahead of their operator (that is, 3−2 becomes 3 2−), so that it can have the numbers ready and waiting for action on a data stack. It turns everything else into nested subroutines; in that way, addresses within addresses can be unraveled in sequence, first onto and then off what is dubbed the return stack. In neither case is time wasted on assigning and finding storage locations.

This setup allows the Forth processor to dedicate separate data paths, each 16 bits wide, to the data stack, the return stack, and the main memory, as well as an I/O bus (Fig. 1). A second I/O bus is 5 bits wide. A single instruction can thus make multiple accesses to memory. With the NC4000A, the stacks are consigned to off-



2. The Forth processor's instructions each consist of a 16-bit op code compiled from one or more Forth words. The first four bit positions define the type of instruction, and the last twelve bits directly control the ALU and registers.

chip high-speed memory. In future versions, however, they may reside on the chip itself.

The Forth processor employs a full 16-bit op code for all instructions. Some of these bits have fixed meanings, others rely on a minimum of decoding in silicon logic. Since its compiler translates the Forth source code directly into these op codes, the programmer need not know the arrangement of bits within each op code in detail. Nonetheless, a grasp of their principle of operation is useful.

A bit at a time

An op code beginning with a 1 indicates an arithmetic instruction or an instruction involving access to main memory or one governing program structure. A 1 followed by three 0s tells the instruction sequencer that the op code represents an arithmetic instruction (Fig. 2) and that it should increment the program counter by one unless the return bit is set.

The next three bits control the function of the ALU, specifying which of seven possible arithmetic and logic actions to perform on inputs T and Y. The T register contains the top element of the data stack; it supplies one of the two inputs to the ALU and also receives information from the ALU. The two following bits specify the source of the other ALU input, Y, as one of the following: the N register, which contains the second stack element; N with carry, used in the ALU operation; the multiplier-divisor MD register; or the square-root SR register. The next bit, T_N, causes the contents of T to be copied into N.

The eleventh bit, marked with a semicolon, is the return bit. This single bit causes a return from subroutine operation to be performed simultaneously with other operations, so that the next instruction executed is actually the next one in the program list. The result is zero time overhead (in most cases) for the return operation.

Instruction construction

As for the last five bits, the one labeled SA indicates that the stack is active; D? specifies a 32-bit shift; the next bit indicates a division-step operation of the ALU; and the last two control the shifter.

Most other Forth instructions also combine several operations. For instance, the familiar Forth word DUP (duplicate top of data stack) combines the two operations, Activate Stack and Copy T into N.

Furthermore, groups of two or more Forth words may be combined into single instructions. The compiler, upon seeing the three-word phrase, OVER SWAP −, does not simply conjoin OVER to SWAP and −, but synthesizes them into a single instruction executed in two simultaneous operations. Specifically, N is fed to the ALU while the ALU is subtracting T from N.

Besides arithmetic steps, the 4000 instruction set includes bit-wise logical instructions; control structure instructions, like jumps and a repeat; data and return stack operators; and various data fetch, data store, and extended address instructions. I/O functions are performed by accessing internal registers.

The Forth processor's instructions correspond either to single Forth words (Table 1) or to multiple Forth words (Table 2). All of the instructions execute in a single clock cycle (except those that access system memory, which require two cycles).

The one-cycle jump to subroutine

Since Forth is optimized for subroutines, the 4000 chip architecture aims at invoking a subroutine with as little overhead as possible. It succeeds, for it executes the equivalent of a subroutine call in the minimum time possible—a single cycle. What's more, in most cases its return from a subroutine takes no time at all.

The Forth processor achieves the single-cycle call because, when the most significant bit of the 16-bit op code is set to 0, it indicates that the remaining 15 bits comprise the address of the subroutine to be called. The chip's instruction sequencer places this address on the address bus in time to latch on the very next cycle the first instruction of the next word. The sequencer's early computation of the next address hinges on its ingenious branch instruction decoding.

The trade-off is, of course, a program space limited to 2^{15} addresses. That is less restrictive than it seems. For one thing, the Forth processor uses word addresses and not byte addresses; the program may occupy 64 kbytes of memory. Given the extreme compactness of the instruc-

Cover: Forth processor chip

Table 1. Instructions corresponding to single Forth words

Stack manipulation		
DUP	(n — n n)	Push copy of top of stack onto stack
DROP	(n —)	Discard top stack element
OVER	(a b — a b a)	Push copy of second (next) stack element onto top of stack
SWAP	(a b — b a)	Reverse order of top two stack elements
Arithmetic and logic		
+	(a b — sum)	Add top two elements as 16-bit two's complement integers
)c	(a b — sum)	Add with carry
−	(a b — a−b)	Subtract top eleMent from second element, as 16-bit two's complement integers
−c	(a b — a−b)	Subtract with carry
OR	(a b — or)	Bit-wise logical OR
AND	(a b — and)	Bit-wise logical AND
XOR	(a b — xor)	Bit-wise logical XOR
2/	(n — n/2)	Arithmetic-shift T register right one bit
2'	(n — n'2)	Arithmetic-shift T register left one bit
0<	(n — ?)	Return true flag (hexadecimal FFFF) if n is negative; otherwise false
D2/	(d — d/2)	Double-length arithmetic-shift right
D2'	(d — d*2)	Double-length arithmetic-shift left
*'	(d — d)	Multiplication step
*−	(d — d)	Signed multiplication step
*F	(d — d)	Fractional multiplication step
/'	(d — d)	Division step
/''	(d — d)	Last division step
S'	(d — d)	Square-root step
Return stack control		
R >	(— n)	Pop top of return stack onto data stack
R@	(— n)	Copy top of return stack onto data stack
#I	(— n)	Copy loop index onto data stack
>R	(n —)	Push top of data stack onto return stack
Structure control		
if		Jump if T register contains zero
else		Jump unconditionally
#loop		Jump and decrement loop counter if it is not zero
times	(n —)	Set repeat instruction counter
call		Jump to subroutine
EXIT		Return
Memory and I/O access		
@	(adr — n)	Fetch value at memory address (2 cycles)
!	(n adr —)	Store value at memory address (2 cycles)
@	(adr — n)	Fetch value at local memory address (2 cycles)*
!	(n adr —)	Store value at local memory address (2 cycles)*
I@	(adr — n)	Fetch value from internal register
I!	(n adr —)	Store value in internal register
n (no name)	(— n)	16-bit literal fetch (2 cycles)
n (no name)	(— n)	5-bit literal fetch*

* Distinguished from preceding instruction(s) by internal structure

tion set, 64 kbytes can contain much more code than in traditional systems. Data buffers, moreover, may be located in the upper (other) 32 kword memory region (since the fetch and store operators use 16-bit addresses), further reducing program size. Finally, Forth is quite amenable to run-time overlay techniques; alternatively, conventional memory-mapping techniques could provide extra program space.

To facilitate a return from a subroutine, called EXIT, another one of the 16 bits in the op code (octal 40) is reserved exclusively for this purpose. Thus the return operation can be specified within the 4000's machine-level op code for simultaneous execution. In a single clock cycle, a word can perform its last operation and return to the word that invoked it.

Of course, since the address bus to main memory is 16 bits wide, the processor can access a full 64 kwords of memory (128 kbytes). Extended address instructions are available to enlarge that to a full 4 Mbytes, through a 5-bit field embedded within fetch and store operations. In addition, the local data fetch and store operations use their 5-bit embedded literal field in the op code to access the first 32 words of main memory in a single instruction, or within two cycles (otherwise it would

Table 2. Instructions corresponding to multiple Forth words

Stack manipulation

SWAP DROP	DROP DUP
SWAP −	SWAP −c
OVER +	OVER +c
OVER −	OVER −c
OVER SWAP −	OVER SWAP −c
OVER OR	OVER XOR
OVER AND	R > SWAP R >
R > DROP	

Full literal fetch

n +	n +c
n −	n −c
n SWAP −	n SWAP −c
n OR	n XOR
n AND	

Short literal fetch

nn +	nn +c
nn −	nn −c
nn SWAP −	nn SWAP −c
nn OR	nn XOR
nn AND	

Data fetch

@ +	@ +c
@ −	@ −c
@ SWAP −	@ SWAP −c
@ OR	@ XOR
@ AND	
DUP @ SWAP nn +	(incrementing fetch)
DUP @ SWAP nn −	(decrementing fetch)

Data store

DUP !	
SWAP OVER ! nn +	(incrementing store)
SWAP OVER ! nn −	(decrementing store)

Extended address data fetch

nn X@ +	nn X@ +c
nn X@ −	nn X@ −c
nn X@ SWAP −	nn X@ SWAP −c
nn X@ OR	nn X@ XOR
nn X@ AND	

Extended address data store

DUP nn X!	

Local data fetch

nn @ +	nn @ +c
nn @ −	nn @ −c
nn @ SWAP −	nn @ SWAP −c
nn @ OR	nn @ XOR
nn @ AND	

Local data store

DUP nn !	DUP nn ! +
DUP nn ! −	DUP nn ! SWAP −
DUP nn ! OR	DUP nn ! XOR
DUP nn ! AND	

Internal data fetch

nn I@ +	nn I@ −
nn I@ SWAP −	nn I@ SWAP OR
nn I@ XOR	nn I@ AND
DUP nn I@ +	DUP nn I@ −
DUP nn I@ SWAP −	DUP nn I@ OR
DUP nn I@ XOR	DUP nn I@ AND

Internal data store

DUP nn I!	DUP nn I! +
DUP nn I! −	DUP nn I! SWAP −
DUP nn I! OR	DUP nn I! XOR
DUP nn I! AND	nn I@!

take four). These can be treated as a set of 32 fast, 16-bit pseudoregisters.

As for actual on-chip registers, the Forth processor has a total of 17, all 16 bits wide and all accessible in one clock cycle. T and N contain the top two elements of the data stack, the elements below being transferred to and from the off-chip data stack. A third contains the two 8-bit stack pointers. The others include two arithmetic registers (multiplier-divisor and square root); the program counter; the return index, or I, register; and a read-only register containing $FFFF_{16}$ for the quick generation of logical true and one that keeps count for the "times" instruction. (In this feature, the instruction currently latched is repeated a specified number of times, freeing the address bus for jobs like fast data transfer.) The remaining eight are four special registers for each of the two I/O ports.

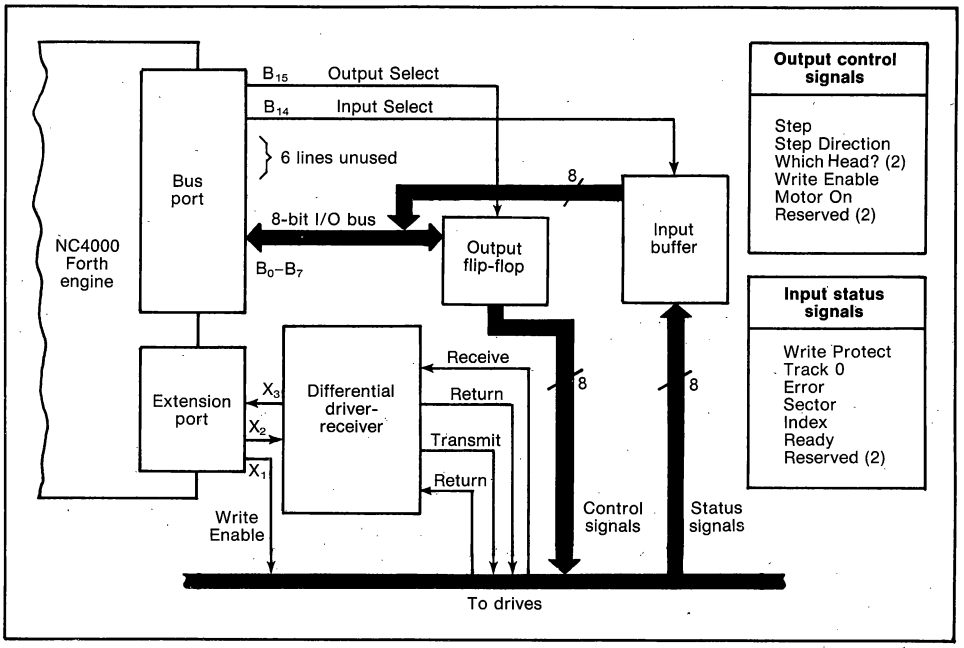
These eight lend a remarkable degree of flexibility to the chip's 16-bit bus port and 5-bit extension port. Each port is represented by a register, and the flow of bits in the ports can be specified by writing to direction registers. Then the output bits can be selected individually by writing to the mask registers. Finally, output bits can also be programmed either to latch on the value last written or to enter a three-state condition after the write operation by having their three-state condition after the write operation by having their three-state registers written to. Port bits marked as input bits can be programmed to yield either the value actually received or the value as compared with an expected value, all in one cycle.

As a final chip feature, a single pin provides one-level interrupt capability, causing execution of interrupt code at octal 40.

In application

One of the goals of the processor is to minimize hardware interfaces. The engine's flexible I/O structure interfaces simply, using very little glue logic, with a variety of chips and buses. And the chip is fast enough to allow functions implemented elsewhere in hardware to be replaced by software. All told, these features make the 4000 versatile enough to fulfill a variety of roles even within a single system. For example, while one serves as the system CPU, another might be the graphics processor and yet another the disk controller.

In this last role, the Forth processor needs only three inexpensive support chips: a differential driver-receiver, an 8-bit flip-flop, and an



3. A Forth processor can readily be programmed to act as a disk drive controller. Because of its highly programmable I/O ports, it needs minimal external logic.

8-bit buffer (Fig. 3). In fact, the flip-flop and buffer could be eliminated if the bus port is not being used for any purpose other than funneling control and status bits. As it is, they prevent stray external voltages from damaging the chip and leave six lines free for other components to be wired onto that 16-bit bus.

In this configuration, the 16-bit bus port is treated as two 8-bit ports. The low-order byte acts like an I/O bus, relaying both output control signals and input status signals. Two of the remaining bits are used as select lines.

The subroutine needed to move the head is simple (see the program, below). Under its guidance, the processor writes output control signals to the drive by selecting the output flip-flop and then writing the appropriate bits to the I/O bus control and status lines. To move the head, the direction bit is set, then pulses are applied to the step line until the head reaches the desired position.

The virtue of brevity

As is typical with Forth code, all the definitions are short. A few words at the beginning of the listing describe the hardware situation and free the programmer from having to bother with such details any further. For instance, the word CONTROL lowers line D_{15} of the bus port, which has been configured to select the output flip-flop on the 8-bit I/O bus. Thus CONTROL is invoked whenever a control signal has to be written to the drive. The word DIRECTION takes a flag from the data stack and writes it to the direction-of-step bit in the control signal. Similarly, STEP writes a 1 or 0 to the step bit.

The remaining definitions then become quite readable. The definitions of INWARD and OUTWARD invoke both CONTROL and DIRECTION. STEP-PULSE cycles the step signal once with an appropriate delay, while STEPS performs a given number of step cycles.

Given these words, the programmer may control the hardware interactively while continuing to develop the more algorithmic sections of the program. For instance, typing OUTWARD 128 STEPS causes the head to move out 128 steps.

Reading data is similarly easy. The processor waits for the sector bit in the status signals to change state, then starts looking for bits at the input. The driver-receiver is connected to the Forth processor's extension port by two lines. Each time a magnetically encoded bit passes under the head, a pulse is sensed on the read line in that port. Appropriate Forth software assembles these pulses into words by reading and shifting.

The Forth processor also proves immediately useful in situations that are complex enough to require the programmability of coded engines but where current processing speeds are insufficient. An example is the simulation of hardware designs in software. In fact, when the Forth processor was designed, it was simulated in Forth software on an LSI-11. □

Program for controlling a disk-drive head

```
: CONTROL
   7FFF DUP   B mask !!
   B !! ;
: DIRECTION ( t=inward — )
   FFBF B mask !!
   B !! ;
: STEP ( t=step motor — )
   FF7F B mask !!
   B !! ;
: INWARD   CONTROL   -1 DIRECTION—;
: OUTWARD   CONTROL   0 DIRECTION—;
: STEP-PULSE   0 STEP   20 DELAY   -1 STEP   20 DELAY ;
: STEPS   ( #pulses — )   1- #DO STEP-PULSE   #LOOP ;
```

NOVIX
10590 North Tantau Avenue
Cupertino, CA 95014
(408) 996-9363
Telex: 352112