MULTIPROCESSOR FORTH
TIME-STATEMENT LEXICON
QUATERNION ROTATIONS
A CHALLENGE OF SORTS
SILICON COMPOSERS

Introduces the
World's First 32-bit Forth Microprocessor:

SC32 Stack-Chip

Also, the SC/FOX Parallel Coprocessor System32,
a PC based development system.

SC/FOX PCS32 (Parallel Coprocessor System32)

SC32 Stack-Chip Hardware Features

* 32-bit CMOS microprocessor, 34,000 transistors.
* One clock-cycle instruction execution.
* Non-multiplexed 32-bit address bus and data bus.
* 16 gigabyte non-segmented data space.
* 2 gigabyte non-segmented code space.
* 8 or 10 megahertz full-static operation.
* Stack depths limited only by available memory.
* Interrupt and interrupt acknowledge lines.
* Bus request and bus grant lines with on-chip tristate.
* Wait state line for slow memory and I/O devices.
* 84-pin ceramic PGA package.
* Unit Price starts at $195.

SC/FOX PCS32 (Parallel Coprocessor System32)

* Uses the SC32 Stack-Chip microprocessor.
* System speed options: 8 or 10 MHz.
* Full-length 8 or 16-bit PC/XT/AT/386 plug-in board.
* 64K to 1M bytes, 0 wait-state static RAM.
* Hardware expansion, two 50-pin strip headers.
* Operates concurrently with PC/XT/AT/386 host.
* Multiple PCS32 board parallel operation.
* PCS32 memory accessible by PC host.
* PCS32 controlled through PC I/O space.
* Data transfer through 16K shared memory window.
* SC/Forth32, interactive Forth-83 Standard with
  32-bit Forth extensions. SC/Forth32 source available.
* Prices start at $1,995 with SC/Forth32.

Ideal for embedded systems control, multitasking and multiple-processor operation, data acquisition,
image processing, or computation intense applications. For additional information, contact us at:

Silicon Composers, Inc., 210 California Avenue, Suite K, Palo Alto, CA 94306  (415) 322-8763
TIME-STATEMENT LEXICON - DAVE EDWARDS

This set of words allows time-based statements in Forth. It constructs code to perform several useful functions, and includes user-definable time units, waiting or flag-testing for a specified period, and more. You may find creative ways to extend this set of commands to define application- or environment-specific functions.

QUATERNION ROTATION CALCULATION
ANTONIO LARA-FERIA and JOAN VERDAGUER-CODINA

A Forth algorithm to directly find the unique axis and the angle of a rotation is presented, a technique that provides some advantage over matrix methods. It is part of the authors' work to apply quaternions in robotics and computer graphics, but quaternions can be used in astronautics, mechanics, robotics, and computer graphics to equal benefit.

MULTIPROCESSOR FORTH KERNEL - BRADFORD J. RODRIGUEZ

This multitasker for a multiple-CPU system handles task allocation transparently, without even requiring the programmer to know how many CPUs are being used. It avoids resource contention, permits intervention via interrupts, and allows triggering of idle tasks without polling or other CPU overhead.

SEARCH ORDER STRUCTURE - CHESTER H. PAGE

The author presents his vocabulary search-order routine, in which FORTH is searched only after all the user-specified vocabularies. It is easy to restore the default condition with a single command. The routine is based on a vocabulary name structure using the dummy link as a pointer to the last word in the vocabulary, and a dummy parameter as a pointer to the dummy link of the next vocabulary to be searched.

THE CHALLENGE OF SORTS

There are all sorts of sorts, and all sorts of programmers. Now the Forth Interest Group—courtesy of major-league efforts by Dennis Ruffer and his on-line cohorts—issues a formal challenge to all Forth programmers. Beat our sort program and have a chance to win a prize. But before breaking out your bubbly, beware! Only the best will rise to the top...
The Forth Interest Group's directors recently decided to distribute public-domain Forth systems on diskette. This is the outgrowth of a years-long debate that has deterred the support of such software, to the organization's goals, to—not least—the vendor community. In earlier times, some feared that public-domain prices would undercut the market of some Forth vendors. Others saw public-domain consumers as (a) entry-level Forth users who will likely graduate to commercial systems in time, and (b) Forth experts who will use and study systems from any source, but who use commercially supported packages when developing programs for sale or under contract.

Today, most Forth vendors provide complete documentation, technical support, and free programming, with which FIG-distributed systems will not and cannot compete. FIG does not offer technical support; users who need such support may use study systems from any source, but who use commercially supported packages when developing programs for sale or under contract.

Today, most Forth vendors provide complete documentation, technical support, consulting, and custom programming, with which FIG-distributed systems will not and cannot compete. FIG does not offer technical support; users who need such support may use study systems from any source, but who use commercially supported packages when developing programs for sale or under contract.

FIG's Mail Order Form is to include F83 v.2.01 and F-PC v.2.25 for IBM PCs and compatibles, and the less-known Pocket Forth for Macintoshes. F83 is familiar to many as the extensive (by comparison with earlier Forths) Forth-83 Model contributed by Henry Luxen and Michael Perry. F-PC is the creation of Tom Zimmer, who has given us a large-environment Forth (files, hypertext, etc., etc.) that deserves lengthy commentary in issues to come. Pocket Forth for the Mac, by Chris Heilman, bills itself as an austere Forth system that follows standard usage and supports toolbox calls and machine code, and accepts text files.

I want to repeat our call for articles about Forth hardware. Last month's editorial gives details about the closing dates, cash awards, etc. Articles can be about a particular Forth chip or board you have used or built, general design philosophy, a survey of entries in the field, you name it. No one with a vested interest in a product is not enough to write about. An event of interest to every programmer is the "Challenge of Sorts" announced in this issue. Dennis Ruffer, head sysop of the GENIE Forth RoundTable, is spearheading this effort to really test your programming skills. The well-designed contest comes complete with prizes and publication for the winners, described later in this issue. The gauntlet has been tossed...

The "Reference Section" continues to grow, with an addition to the on-line resources and two new categories: ANS Forth lists X3J14 representatives who are willing to take your proposals and concerns about the developing ANS Forth directly to that committee; and Forth Instruction provides a place to find ongoing educational interests of interest. Send any additions and elaborations to us.

(EDITORIAL continued on page 37)
Object Commentary
Dear Mr. Ouverson,

This is a comment on Mike Elola’s comment on my comment on his object-oriented article in Forth Dimensions (X/5)!

First, I’m glad that my comment has stirred so much enthusiasm. I have received many positive letters on it. I hope this interest in object-oriented Forth (OOF) continues to grow.

Using the now-famous arithmetic average example, listed below, Elola suggested that the phrase `USE FLOAT` should not be necessary in an OOF language.

```
USE FLOAT
A @ B @
+ 2/
C !
```

CSU Forth is not an OOF language. I found no reason to make it so. Instead, it accommodates OOF principles. The original spirit of Forth, as I understood it, is program freedom and language extensibility. I therefore designed CSU Forth to be 100% compatible with the standards, yet to offer the programmer an excellent way of program design if he/she wants to use it. In CSU Forth, classes and objects are Fort language words. The way they do what they do is found in their `DOES>` parts.

In the arithmetic average example, the phrase `USE FLOAT` is needed for the methods `+` and `/`, not for the float objects `A`, `B`, or `C`. These methods will behave differently in a float class than in an integer class. Using `USE`, the programmer doesn’t have to create a special object to call these methods. It also allows the programmer to use familiar word names like `2/` without binding ambiguity. The same technique is found in standard Forth whenever you invoke a vocabulary. Perhaps writing the example as below will remove this misunderstanding:

```
A @ B @
USE FLOAT
+ 2/
USE FORTH
C !
```

Finally, I hope that the next Forth standard will not be called “Forth++.” Forth-90 is what I’d like to see. I especially pray that if the ANSI committee decides to implement OOF, they will not be inspired by the cryptic and complex standards of C++. OOF principles are much simpler and easier to implement than some folks will lead you to believe.

Sincerely,
Ayman Abu-Mostafa
7932 Lampson Ave. #25
Garden Grove, California 92641-4147

Conditional-Stack Caveat
Dear Marlin,

There is a serious misunderstanding of the `IF ... ELSE ... THEN` and `IF ... THEN` constructs in Abu-Mostafa’s article on branchless conditionals (“Forth Needs Three More Stacks,” FD XI/1). The standard interpretation of these constructs includes the following points:

1. IF removes a single value from the parameter stack and processes it as a Boolean.
2. If the Boolean value is true, processing continues with the words immediately following IF and continues up to a matching ELSE or THEN.
3. If the Boolean value is false, the words immediately following IF are skipped up to a matching ELSE or THEN, and processing is resumed at the matching ELSE or THEN.
4. The group of words that is skipped has no effect on the parameter stack, the return stack, or any variable in the dictionary, no matter how long or complicated the group of words may be.

In these rules, “matching” means that any `IF ... ELSE ... THEN` or `IF ... THEN` constructs that are nested inside in skipped code must be passed over, and their ELSE or THEN parts ignored.

One might quibble about whether my wording of the rules is precise, complete, or the most concise possible, but the intent is clear to all users of Forth. In particular, skipped words must not affect the parameter stack. Abu-Mostafa’s fourth rule of processing his proposed condition stack does have skipped IF words affecting the parameter stack. This is wrong.

One can implement branchless conditionals correctly without the use of a condition stack. Instead, one needs only an execution control flag (ECF) and a nesting depth counter (NDC). During initialization of the system, ECF is set true and NDC is set to zero; then the following rules apply:
1. As each word is parsed from the input stream, ECF is examined. The word is executed if ECF is true, and is skipped if ECF is false.
2. Execution of IF causes the top value on the parameter stack to be stored into ECF.
3. Execution of ELSE causes false to be stored into ECF.
4. Execution of THEN is a no-operation.

During "skipping," special actions are taken if any of the words IF, ELSE, or THEN are encountered, as follows:

1. Skip over IF causes NDC to be incremented.
2. Skip over ELSE causes a test of NDC, and
   2a. If NDC is zero, true is stored into ECF.
   2b. If NDC is non-zero, no action is taken.
3. Skip over THEN also causes a test of NDC, and
   3a. If NDC is zero, true is stored into ECF.
   3b. If NDC is non-zero, NDC is decremented.

The use of the NDC makes the "skipping" state pass over matching pairs of IF and THEN words without ending the skipping state. It works up to a nesting level equal to the overflow count of NDC. (Probably larger than the storage allocation of any possible condition stack!) The algorithm is essentially the same as one for finding matching left and right parentheses in algebraic expressions. Evaluation of an algebraic expression is nicely done using a stack to hold intermediate results, but the stack is quite unnecessary if one is only interested in finding matches of parentheses.

In addition to the serious misstatement of IF ... THEN processing, there are other less serious problems with Abu-Mostafa's branchless conditionals:

- Marking words for special processing by setting a bit in the NFA is only satisfactory for interpretive mode. If it is used on compiled code, it would require the outer interpreter to execute >LINK for every word that is being skipped during a skip sequence. This would surely be very slow.
- DO ... LOOP still requires a backward branch. Some of the words that Abu-Mostafa hopes to eliminate with his branchless conditionals will still be required to implement this backward branch.
EVER since my first process-control program, when I needed to execute code on a timed basis, I have been working on a lexicon for time statements in Forth. Consulting my back copies of *Forth Dimensions*, I discovered a typically excellent article by William Ragsdale (issue V15) which developed syntaxes of the type:

```
TICK IF EACH-SECOND
TOCK IF EACH-MINUTE
THEN THEN
```

It was perfect for my needs in that first control program. My second program required more sophistication than just each-second and each-minute kinds of statements, so I began extending Ragsdale's basic ideas into the present lexicon.

The lexicon allows a variety of time-based statements in Forth. It does not use an operating-system approach (in which the "system" maintains a set of timers available for use), but constructs in-line Forth code which performs the timing functions. The lexicon allows syntaxes for dealing with a variety of time functions, including:

- A declarable set of time units (milliseconds, seconds, minutes, etc.).
- Perform processes on each new time unit (similar to Ragsdale's TICK TOCK system).
- Monitor a time lapse.
- Declare a time period.
- Detect whether a period has elapsed.
- Wait to proceed until after a specified period.
- Monitor a condition (flag) for a time period.

Functions required beyond this set can easily be expressed in phrases using the core lexicon words.

The first design requirement was that the lexicon be able to handle multiple time units: milliseconds, seconds, minutes, even hours and days. To implement this, a set of entities was created that is collectively called the time units, which are simply numbers used by the code to distinguish the current time unit (TU).

Words in the lexicon are designed to run repeatedly inside a Forth control structure for the period of interest. Time lapses are measured by keeping a note of the value of the time units on the previous pass through the loop and comparing it to the value of the time unit on the current pass through the loop. While in the loop, the words need access to various parameters:

- A counter plus new's parameters (c v TU).
- To detect if a period has elapsed requires: the total period plus lapse's parameters (P c v TU).

The first word in the lexicon is @TIME, which returns the value of the specified time unit. It is used by most words in the lexicon and is shown in Figure One-a. Its definition is implementation-specific and is discussed in the implementation section.

It can be used directly to return the value of particular time units, for instance:

```
: @SECS SECS @TIME ;
: @MINS MINS @TIME ;
```

e tc.

NEW expects the previous value of a time unit, along with the time unit, on the stack. This word is analogous to Ragsdale's TICK but has been generalized to handle the different units. It leaves an updated TU value and a flag that is true if there was a new value, false otherwise. (See Figure Two.)

Notice that NEW actually leaves the difference between the two time unit values on the top of the stack, not a pure flag. This allows NEW to be reused in the definition of LAPSE.

NEW can be used immediately in a Forth control structure to execute EACH-time unit functions similar to Ragsdale's original system. For example:

```
MINS @TIME
SECS @TIME
```

```
BEGIN
SECS NEW
IF EACH-SEC
THEN SWAP
```

THE time units are independent of the timing basis.

- the current time unit (TU)
- the value of the time unit on the last pass (v)
- a counter to accumulate the time lapse (c)
- the specified time period or limit value (P)

All of these parameters are passed on the stack and they remain on the stack for the duration of the loop. In this sense, the design uses stack data structures—there is a different data structure for the various aspects of time being monitored:

- To access a time value requires: the time unit (TU).
- To detect a new value requires: the time unit and a previous value (v TU).
- To monitor a time lapse requires:
Figure One. Implementation-dependent internals.

Figure One-a
: @TIME ( TU -- v )
  CASE
  mSEC OF code to fetch value of milliseconds ENDOF
  SECS OF code to fetch value of Seconds ENDOF
  etc. ENDCASE ;

Figure One-b
: MAX-TU ( TU -- n )
  CASE
  mSEC OF 1000 ENDOF
  SECS OF 60 ENDOF
  MINS OF 60 ENDOF etc. ENDCASE ;

Figure One-c
: MAX-TU ( TU -- n )
  CASE
  mSEC OF 10 ENDOF
  cSEC OF 10 ENDOF
  dSEC OF 10 ENDOF
  SECS OF 60 ENDOF etc. ENDCASE ;

Figure One-d

Physical Timers: VARIABLE msec contains milliseconds
VARIABLE mins contains minute number

Time-Units:  mSEC cSEC dSEC SECS MINS HRS
: @TIME CASE
  mSEC OF msec @ ENDOF
  cSEC OF msec 10 / ENDOF
  dSEC OF msec 100 / ENDOF
  SECS OF secs @ ENDOF
  MINS OF mins 60 / ENDOF etc. ENDCASE ;

Figure Two. Basic lexicon.

: NEW ( v TU -- v' f \ True if v' not equal to v )
  @TIME DUP ROT - ;

: CLK-ON ( TU -- c=0 v TU \ Clears a LAPSE counter )
  0 SWAP @TIME SWAP ;

: LAPSE ( c v TU -- c' v' TU \ increments c by time lapse )
  DUP >R NEW DUP
  0< IF R@ MAX-TU + THEN SWAP >R + R> R> ;

: ELAPSED ( P c v TU -- P c' v' TU f \ True if c' >= P )
  LAPSE 3 PICK 3 PICK > ;

Figure Three. Control structures.

: PERIOD
  COMPIL   CLK-ON
  [COMPIL] BEGIN
  COMPILE ELAPSED ;   IMMEDIATE

: TIME
  COMPIL   OR
  [COMPIL] UNTIL
  COMPILE 2DROP ;   IMMEDIATE

Note that current values of the time units are always on the stack during the
BEGIN ... UNTIL structure.

CLK-ON sets up the stack values for a
time LAPSE monitor—it initializes the
lapse count (c) to zero and runs @TIME to
provide an initial time unit value. (See
Figure Two.)

LAPSE expects a single-precision
count, along with NEW's parameters on the
stack. The count is incremented by the time
lapse between the previous value and the
current value, and is shown in Figure
Three.

The word MAX-TU in the definition of
LAPSE is a word which pushes the maxi-
mum value of any TU in the system, and is
used to handle wrap-around of the value.
For instance, when using SECONDS as the
time unit, if the previous value was 59 and
the current value is two, then wrap-around
has occurred and the MAX-TU for SEC-
ONDS (60) must be added to the result left
by NEW (-57). MAX-TU is discussed in
detail in the implementation section.

LAPSE can, therefore, be run only
slightly more frequently than the next
higher time unit in the system—if SECS is
the current time unit, then LAPSE need
only be run once every 59 seconds or so.

LAPSE can be used in a variety of ways.
The following phrase leaves a number on
the stack which indicates how long it took
before the flag-leaving Forth phrase went
true:

mSEC CLK-ON
BEGIN LAPSE ... ( f )
UNTIL 2DROP ( n )

The next construct will remain in the loop,
running the code between WHILE and RE-
PEAT for ten minutes:

MINS CLK-ON
BEGIN
  LAPSE 3 PICK 10 <
  WHILE ... REPEAT
  2DROP DROP

Forth Dimensions  8
Note again that care must be taken in the
Forth phrases between WHILE and RE-
PEAT, as there are timing-control numbers
on the stack for the duration of the loop.

ELAPSED expects a total-period,
single-precision number below
a word which simply consumes the speci-
fied period of time. We have named this
count, value, time unit) are simply dropped
at the end of the loop, being of no further
ments of the form:

This construct:

CLK-ON BEGIN ELAPSED (condition)
OR UNTIL

has come to be so useful, and was used so
often in process-control code, that compiling
words have been defined to build it. The
phrase CLK-ON BEGIN ELAPSED is built
by the word PERIOD, and the phrase OR
UNTIL 2DROP is built by the word TIME.
Their definitions are given in Figure Three.

This control structure, in conjunction
with the subsequent tests, allows state-
ments of the form:

10 SECS PERIOD
?HEATED
TIME 2DROP
(to simply proceed)

20 SECS PERIOD
?TERMINAL
TIME > IF
"Key struck within 20 secs"

THEN

Optional control structures.

: MONITOR (COMPILE) TIME
  COMPIL 2DROP ; IMMEDIATE

: DETECTED (COMPILE) TIME
  COMPIL > ; IMMEDIATE

: TIMED (COMPILE) TIME
  COMPIL SWAP
  COMPIL OVER
  COMPIL > ; IMMEDIATE

The values used during the loop (period,
count, value, time unit) are simply dropped
at the end of the loop, being of no further
interest in this particular case.

Another common requirement is to wait
for a specified total time period and simul-
taneously monitor some condition. If the
condition goes true, the loop is left immedi-
ately; otherwise, the total specified period
is consumed. This can be accomplished
with the phrase:

10 MINS CLK-ON
BEGIN ELAPSED ... ( f )
OR UNTIL 2DROP ( P C )

This construct leaves two numbers on
the stack, the original limit specified for the
period and the lapsed time when the loop
terminated.

If the period timed out (the condition
did not go true in the specified time period),
10 MINS PERIOD
?ALARM
TIME SWAP OVER
> IF ." Alarm after "
ELSE ." No alarm within "
THEN ." minutes"

If desired, optional compiling utilities shown in Figure Three can be defined, allowing statements like:

20 SECS PERIOD
?TERMINAL DETECTED
IF
." Key struck within 20 secs"
THEN

10 MINS PERIOD
?ALARM TIMED
IF ." Alarm after "
ELSE ." No alarm within "
THEN ." minutes"

Implementation

The lexicon expects values to which it has access to be changing automatically on a timed basis. The physical source of these locations (whether they are created by a hardware interrupt and interrupt code or by a real-time clock) is immaterial to the design.

The steps to implement the timing system are:
1. Decide on the set of time units.
2. Write @TIME to interface to the hardware-specific timing information.
3. Write MAX-TU for the set of time units chosen.
4. Use the high-level definitions for the remaining words.

The time units are usually defined in Forth as constants or variables. There is no restriction on the actual values, except that they must be distinct from one another. The interface to the time units is formalized through two words:

@TIME fetches the value of the named time unit.
MAX-TU pushes the maximum (i.e., wrap-around) value of the named time unit.

Note: MAX-TU is only used in the word LAPSE, so it does not need to be separately defined; it can be written in-line in the definition of LAPSE. It is created here simply to clarify the explanation of the required functions.

The structure of @TIME is shown in Figure One-a. It uses the time unit on the stack as the input to a CASE statement, the clauses of which perform the particular fetch operation and form part of the machine- and even the application-dependent part of the code.

The structure of MAX-TU is shown in Figure One-b. It again uses the TU as the input to a CASE statement and pushes the wrap-around value of the current time unit depending on the set of time units chosen in the implementation, in this case msec, secs, and mins.

As a further example, if the required set of time units included, say, msec and tenths of a second (dsec), then MAX-TU would be as shown in Figure One-c.

The chosen set of time units is independent of the physical timing basis. For instance, if a one millisecond interrupt was used to provide variables for milliseconds (msec), seconds (secs), and minutes (mins), it is still possible to implement a set of time units that includes more than just msec, secs, and mins. This is illustrated in Figure One-d.

Example Implementations

Interrupt-based Timing

Basis of timing: 20 millisecond (50 Hz) interrupt
Variables: Ticks, Seconds, Minutes, Hours

Interrupt code:

increments Ticks
if Ticks > 49, clear Ticks
increment Seconds
if Seconds > 59, clear Seconds
increment Minutes
if Minutes > 59, clear Minutes
increment Hours

In such a system, the allocation of the time units is most conveniently handled by using the VARIABLE address as the CONSTANT of the time units:

Ticks CONSTANT TICKS
Seconds CONSTANT SECS
Minutes CONSTANT MINS
Hours CONSTANT HRS

In effect, no redefinition is needed here—the names of the variables can act as the time units in this particular implementation. Having made this decision, the word @TIME is simply Forth's @ (fetch):

: @TIME @ ;

and MAX-TU is:

: MAX-TU
DUP TICKS =
IF DROP 50
ELSE HRS =
IF 24 THEN
THEN ;

Real-time Clock

The Motorola MC 14 6818 Real Time Clock (RTC) has registers which constantly contain the values of:

seconds (reg 0)
minutes (reg 2)
hours (reg 4)

Given these physical addresses, a natural allocation of the time units is:

0 CONSTANT SECS
2 CONSTANT MINS
4 CONSTANT HRS

A word called @RTC is developed to fetch from an RTC register; it expects the register number on the stack and returns the value of that register:

: @RTC ( r -- n )
( code to return the value of )
( Real-Time Clock register r )
;

With this definition in place, @TIME can be defined as:

: @TIME @RTC ;

and MAX-TU can be defined as:

: MAX-TU
DUP HRS =
IF DROP 24

(Continued on page 30)
This program presents the advantage of using quaternions, instead of matrix methods, to calculate rotations. A Forth algorithm to find the unique axis and the angle of a rotation is presented. It is part of a work to apply quaternions in robotics and computer graphics.

According to reference [1], quaternions require fewer mathematical operations than matrix methods. An additional feature of quaternions is that they give the axis and the angle of a rotation directly.

Quaternions can be applied in many areas, for example astronautics [2], mechanics [3], robotics [4,5,6,8], and computer graphics [1,7]. The program presented here focuses only on the use of quaternions to calculate rotations.

---

Written in PC/FORTH V3.1 from Laboratory Microsystems. Uses code (SIN & COS routines, plus data tables) from LM's utility file FORTH.SCR, which are in screens # 5, # 6.

(C) JVC all but Screens # 5 & # 6.

---

### Numbers and Precision

The version of Forth-83 used is Laboratory Microsystems, Inc. (LMI) PC/Forth 3.10.

The program was written using straightforward, single-length arithmetic. The reasons for doing this are:

- In LMI's version of Forth, no words are provided for multiplying or dividing double-length integers.
- Even though the above-mentioned language can work with an 8087 coprocessor—thus allowing the use of floating-point arithmetic, no 8087 was present on the equipment used to develop this program.
Anyhow, the program can easily be changed to work with floating-point precision simply by entering LMI’s Forth editor and changing all single-length arithmetic words to the corresponding floating-point operators.

Overflow and Inexact Results

Since single-length arithmetic has been used, depending on the data the user feeds the program, it may give erroneous results due to internal overflow. Care should be taken to avoid such a situation; sometimes, results shown as the negative of certain values can indicate an internal overflow (e.g., since 32767 is the greatest signed number that can be represented, an overflow-bound sequence like 32767 1+ . would yield -32768).

On the other hand, when there is no overflow the results may be slightly incorrect due to the poor precision provided by 16-bit signed integer operations.

The magnitude of the two possible errors mentioned above will increase as more and more rotations are performed upon one single vector. In fact, the reasonable maximum number of rotations in such cases turns out to be two.

Extra Code

Some of the words contain code that is not being used by the main RUNME word or the words that it calls. That code expresses programming alternatives; some of the routines and the ideas they represent can be used to change or enhance the program.

How to Run the Program

After entering LMI’s PC-Forth, the disk drive containing the screen file QUATERN.SCR should be specified to the system, i.e.:

```
USING <DRIVE>:QUATERN.SCR
```

When the file has been located and acknowledged by PC/Forth, load the program by entering:

```
RUNME.
```

To execute the program, simply type the word RUNME.

Note: The program uses PC/Forth’s assembler in the SIN and COS routines, so the file ASM86.BIN should be present on the PC/Forth disk. Otherwise, the program won’t be loaded.
Bibliography


Joan Verdaguer-Codina works in the Centre d’Alt Rendiment, a high-performance sports center in Catalonia.
This article describes a Forth multitasker for a multiple-CPU 68000 system. This multitasker:

- automatically distributes the task load among the available processors, without explicit effort by the programmer;
- provides a means to prevent conflicts when different tasks or different CPUs attempt to use the same resource;
- allows tasks to sit in an idle state, awaiting an external trigger, without polling or other CPU overhead;
- allows interrupts to alter the scheduling of tasks.

The principles described herein can be applied to multiprocessor systems using other CPUs, and even to single-processor systems.

For more throughput, plug in another CPU!

The Application

The multiprocessor kernel was originally developed for a performance-lighting control system. The processing demands of this system were quite strict, and fell into three categories:

- Event-driven processing—initiated by external events, such as the system operator moving a control handle. Requires a response time on the order of 100 milliseconds (msec).
- Time-driven processing—must occur at periodic intervals. Most of this repetitive processing occurs every 40 msec, but intervals from ten msec to 1000 seconds

### Listing One

```
0 \ ******************** MULTIPROCESSOR TASKER v3 04 02 06 BJR *******
1 4 CONSTANT CELL
2 : CELLS \ n \ n 4 * ;
3
4 \ SUBROUTINE \ VARIABLE \ 4 ALLOT \ [COMPILE] ASSEMBLER \ ; \ Fig
5 \ : SUBROUTINE CREATE \ [COMPILE] ASSEMBLER ; \ Forth-83
6
7 181 LOAD \ task area definition
8 182 LOAD \ internal data areas
9 183 LOAD \ tasker subroutines
10 189 LOAD \ tasker primitives
11 190 LOAD \ defining words & initialization
12 191 LOAD \ task setup
13
14
15
```

```
0 \ Task area structure 11 02 86 BJR
1 \ Offsets into the task area
2 HEX 80 CELLS CONSTANT USIZE \ size of user variables
3 0 CONSTANT UAREA USIZE + \ user variables
4 DUP CONSTANT RSTACK 80 CELLS + \ return stack
5 DUP CONSTANT RTP 80 CELLS + \ parameter stack
6 DUP CONSTANT RTP 4 CELLS + \ top safety margin
7 CONSTANT TASKSIZE \ total size of the task area
8 DECIMAL
9 10 \ offsets (from UAREA) to selected user variables
10 2 CELLS CONSTANT +RP-TEMP 3 CELLS CONSTANT +S0
11 4 CELLS CONSTANT +RO 5 CELLS CONSTANT +TIB
12
13 \ TASK \ VARIABLE \ TASKSIZE CELL \ ALLOT \ ; \ Fig
14 \ TASK CREATE TASKSIZE ALLOT ; \ Forth-83
15
```

```
0 \ Semaphore queues structure ( 30 7 06 BJR 15:30 )
1 : DECIMAL
2 8 CONSTANT .SEMA 10 CONSTANT .IBIT \ semaphore field offsets
3
4 \ : SEMAPHORE \ VARIABLE \ 2 CELLS ALLOT \ ; \ Fig
5 \ : SEMAPHORE CREATE \ 3 CELLS ALLOT \ ; \ Forth-83
6
7 SEMAPHORE READYQ \ ready queue header, 3 cells
8
9 HEX CB USER SELFQ \ "self-queue" to suspend tasks
10 \ 3 cells in task's user area; actual offset is system dependent
11
12 \ USER MYTASK \ returns addr of currently-executing task!
13 SELFQ MYTASK \ CONSTANT +SELFQ \ offset from base of task area
14 \ 15
```

Forth Dimensions 14 Volume XI, Number 3
are possible.
- Background processing—continuous processes, such as display updates, which have relaxed timing requirements.

Systems whose capacity varied over a ten-to-one range were to be sold. So that the processing power could be configured to suit, the architecture was designed (Figure One) to use from one to four 68000 processors, with a common memory of CMOS RAM. The processors each had private EPROMs for program storage.

The programmer needn't know how many CPUs are installed.

Requirements of the Multitasker
The 68000s were to be programmed in multitasking Forth. Our original software design assumed a round-robin task; however, we soon switched to a more "traditional" queued tasker for several reasons:
1. We expected to have a large number of idle tasks—awaiting some external or timed event—at all times. Our studies for this configuration indicated that, if 80% or more of the tasks were idle, the queued tasker is more efficient. (If fewer than 80% are idle, it is faster to poll them round-robin than to move them on and off a "ready queue."

2. We needed to guarantee order of service. Our resources—particularly communications—required that competing requests be serviced on a first-come, first served basis.

3. Curiosity. We hadn't seen a queued tasker in Forth, and wanted to see how easily it could be done!

Queue Storage
Figure Two shows the structure of a task queue. Each queue is stored as a singly linked list. The link is stored in the first cell of each task's user area, looking very much like a typical round-robin multitasker. The differences are:
- there are many linked lists,
- they are not linked in a circle, and
HS/FORTH's compilation and execution speeds are unsurpassed. Compiling at 20,000 lines per minute, it compiles faster than many systems link. For real jobs, execution speed is 79 Standard unsurpassed. Even non-optimized programs run as fast as ones produced by most C compilers. FORTH systems designed to fool benchmarks are slightly faster on nearly empty do loops, but bog down when the colon nesting level approaches anything useful, and have much greater memory overhead for each definition. Our optimizer gives assembler language performance even for deeply nested definitions containing complex data and control structures.

HS/FORTH provides the best architecture, so good that another major vendor "cloned" (rather poorly) many of its features. Our Forth uses all available memory for both programs and data with almost no execution time penalties, and very little memory overhead. None at all for programs smaller than 200kB. And you can reuse segments anytime, without a system reigon. With the GigaForth option, your programs transparently enter native mode and expand into 16 Meg extended memory or a gigabyte of virtual, and run almost as fast as in real mode.

Benefits beyond speed and program size include word redefinition at any time and vocabulary structures that can be changed at will, for instance from simple to hashed, or from 4 bytes to 16 bytes. You can reheader word names and reclaim space at any time. This includes automatic removal of a colon definition's local variables.

Colon definitions can execute inside machine code primitives, great for interrupt & exception handlers. Ml-merllt words are easily implemented. And code words become incredibly powerful, with multiple entry points not requiring jumps over word fragments. One of many reasons our system is much more compact than its immense dictionary (16000 words) would imply.

INCREDIBLE FLEXIBILITY

The Rosetta Stone Dynamic Linker opens the word of utility libraries. Link to resident routines or link & reroute routines interactively. HS/FORTH preserves relocalatability of loaded libraries. Link to BTREIEVE METAWINDOWS HALO HOOPS ad infinitum. Our call and data structure words provide easy linkage.

HS/FORTH runs both 79 Standard and FORTH 83 programs, and has extensions covering vocabulary search order and the complete FORTH 83 test suite. It loads and runs all FIG Libraries, the main difference being they load and run faster, and you can develop larger applications than with any other system. We like source code in text files, but support both file and sector mapped Forth block interfaces. Both line and block file loading can be nested to any depth and includes automatic path search.

APPLICATION CREATION TECHNIQUES

HS/FORTH assembles to any segment to create stand alone programs of any size. The optimizer can use HS/FORTH as a macro library, or complex macros can be built as colon words. Full forward and reverse labeled branches and calls complement structured flow control. Complete syntax checking protects you. Assembler programming has never been so easy.

The MetaCompiler produces threaded systems from a few hundred bytes, or FORTH kernels from 2kB. With it, you can create any threading scheme or segmentation architecture to run on disk or ROM. You can turnkey or seal HS/FORTH for distribution, with no royalties for turnkeyed systems. Or convert for ROM in saved, sealed or turnkeyed form.

HS/FORTH includes three editors, or you can quickly shell to your favorite program editor. The resident full window editor lets you reuse former command lines and save to or restore from a file. It is both an indispensable development aid and a great user interface. The macro editor provides reusable functions, cut, paste, file merge and extract, session log, and RECOMPILE. Our full screen Forth editor edits file or sector mapped blocks.

Debug tools include memory stack dump, memory map, decompile, single step trace, and prompt options. Trace scope can be limited by depth or address.

HS/FORTH lacks a "modular" compilation environment. One motivation toward modular compilation is that, with conventional compilers, recompiling an entire application to change one subroutine is unbearably slow. HS/FORTH compiles at 20,000 lines per minute, faster than many languages link — let alone compile! The second motivation is linking to other languages. HS/FORTH links to foreign subroutines dynamically. HS/FORTH doesn't need the extra layer of files, or the programs needed to manage them. With HS/FORTH you have source code and the executable file. Period. "Development environments" are cute, and necessary for unnecessarily complicated languages. Simplicity is so much better.

HS/FORTH Programming Systems

Lower levels include all functions not named at a higher level. Some functions available separately:

- Documentation & Working Demo
  - (5 books, 1000 + pages, 6 lbs) $ 95
  - Student $145
  - Personal optimizer, scaled & quad integer $245
  - Professional 80x87 assembler, turnkey $395
  - dynamic strings, multitasker, RSDL linker, physical screens $495

- Production ROM, MetaCompiler, Metawindows $495
- Level upgrade, price difference plus $ 25
- OBJ modules $435
- Rosetta Stone Dynamic Linker $ 95
- Metawindows by Metagraphics (includes RSDL) $145
- Hardware Floating Point & Complex $ 95
- Quad integer, software floating point $ 45
- Time slice and round robin multitasker $ 75
- GigaForth (80286/386 Native mode extension) $295

HARVARD

SOFTWORKS

PO BOX 69
SPRINGBORO, OH 45066
(513) 748-0390
1 \ get current task from ready queue
2 READYQ \ AR0 ,L MOV,
3 700 \ SR .W OR, .BIT AR0 & .B TAB, MI HERE 4- ++ BCC, \ i
4 1 \SEMA AR0 & .W ADDQ, \ increment semaphore
5 BT IF, 0 \ TRAP, (ready queue empty!) THEN,
6 AR0 I U .L MOV, U I AR0 .L CMP, \ get head of readyq
7 EQ IF, AR0 4 AR0 & .L MOV, THEN, U I AR0 .L MOV,
8 7 \ .BIT AR0 & .B BCLR, \ release queue
9 \ make semaphore's task current
10 8 U & IP .L MOV, RP I+ 5 \.L MOV, RP I+ IP .L MOV, \ restrict
11 \ Resume execution
12 THEN, DR7 SR .W MOV, RTS, \C \ restore interrupt level
13 --> uses dr7,ar2 expects u,sp,rp valid

0 \ Tasker -- (wait) (AA000 21 7 86 BJR 23:45)
1 SUBROUTINE (NEXT) NEXT ;C \ start forth inner interpreter
2 DECIMAL 15
3
4 (SIGNAL) and (WAIT) assume that all context has been stacked,
5 and that the last thing stacked is the PC for the restore.
6 This is normally accomplished by entering via JSR.
7 8 This subroutine is made the starting PC of a newly-initialized
9 task. When the new task is started from a queue, its IP and SP
10 will be unstacked, and then (NEXT) will be entered...starting
11 high-level execution at the given IP.
12 13
14
15

0 \ Tasker -- start PAUSE (AA000 30 7 86 BJR 17:32)
1 CODE START S I+ AR1 .L MOV, \ tadr -- start new task
2 (START) ++ BSR, NEXT ;C
3 CODE SIGNAL S I+ AR0 .L MOV, \ qadr -- acquire resource
4 (SIGNAL) ++ BSR, NEXT ;C
5 CODE WAIT S I+ AR0 .L MOV, \ qadr -- acquire resource
6 (WAIT) ++ BSR, NEXT ;C
7 CODE PAUSE READYQ \ AR0 .L MOV, \ --: switch to next task
8 (SIGNAL) ++ BSR, NEXT ;C
9 CODE SUSPEND SELFQ MYTASK - U | AR0 LEA, --: suspend self
10 (WAIT) ++ BSR, NEXT ;C
11 CODE RESUME S I+ AR0 .L MOV, \ taskadr -- resume task
12 SELFQ MYTASK - # AR0 ADD, (SIGNAL) ++ BSR, NEXT ;C
13 \S
14 Note that the forth context information(S,IP,RP) is saved by
15 the task switching primitives.

0 \ Tasker -- newdevice - newresource (30 7 86 BJR 15:43)
1 HEX
2 : NEWDEVICE \ qadr --: initialize semaphore to 0 for event
3 DUP DUP ! DUP DUP CELL +! 0 SWAP 2 CELLS +! ;
4 : NEWRESOURCE \ qadr --: init. semaphore to 1 for shared resou
5 DUP NEWDEVICE 10000 SWAP 2 CELLS +! ;
6 8 DECIMAL \S
7 9
8

- tasks are constantly being moved (i.e., relinked) from one list to another—the task order is dynamic, rather than static.

This approach involves a minimum of data movement. A task can be moved from one queue to another by changing four links.

The linked list requires a very small memory overhead—three cells (12 bytes) per queue. This “queue header” contains a head pointer, a tail pointer, a 16-bit integer semaphore, and a multiprocessor “lock” bit.

A snapshot of the queues during execution might look something like Figure Three.

**Allocation of Tasks**

Tasks which are ready to run are held on a ready queue. When a CPU finishes one task—perhaps by executing PAUSE—it will pick up the next task from the head of the ready queue.

All the CPUs pick up tasks from the same ready queue, so the first CPU to become available will service the first waiting task. Since all CPUs see the same memory and I/O space, and have identical copies of the program, any task can run on any CPU.

This means that the programmer does not need to know which CPU his code is to run on. In fact, the programmer does not need to know how many CPUs are installed. The task load is automatically divided among the installed CPUs. For more throughput, plug in another processor!

The limiting factor is bus contention. We minimized this by giving each CPU a private (but identical) program memory, but still the CPU becomes saturated when three or four CPUs compete for data memory.

**Protection of Shared Resources**

Our critical resources were protected against conflicting access with the classic “semaphore” operators, WAIT and SIGNAL. Most textbooks on operating systems describe these in detail, so this will be just an overview.

Each protected resource has an integer semaphore. Its initial value, +1, indicates that the resource is available. A zero semaphore means the resource is in use. A negative value, -N, indicates that it is in use and that there are N pending requests for the resource.

Whenever a task requests a busy re-
source, it is placed on a "wait queue" for that resource. This queue is first-in, first-out—or, more to the point, first-come, first-served. Tasks on a wait queue consume no CPU time.

The programmer does this through the operators WAIT and SIGNAL.

**WAIT** decrements a given semaphore. If the resource is busy, the task is parked on the wait queue, and a new task is started from the ready queue (Figure Four).

**SIGNAL** increments the semaphore. If a task is waiting for this resource, pause the current task and start the waiting task (Figure Five).

**WAIT** and **SIGNAL** surround the code which uses the protected resource, as follows:

**SEMAPHORE DISK**

: xxx DISK WAIT
    code to access disk
    DISK SIGNAL ;

Note that we can always tell the state of the resource and its wait queue by examining the semaphore value. In our implementation, the semaphore and the header for the wait queue are stored together (Figure Two).

(For those familiar with the "monitor" construct used in many concurrent languages: monitors can be implemented very easily with semaphores. Each monitor requires one semaphore, and all routines in the monitor WAIT and SIGNAL that semaphore.)

**WAIT** and **SIGNAL** are required to be indivisible. Nothing must alter or use the semaphore and queue data structure while a WAIT or SIGNAL is in progress. In a single-CPU system, this is done by disabling interrupts. In a multiple-CPU system, we must further guard against, say, two processors WAITing the same semaphore simultaneously.

In the 68000, this is done with the indivisible TAS (Test And Set) instruction. This instruction is not powerful enough to use in place of semaphores, but it is sufficient to protect the semaphores themselves from conflicting access. Figure Six shows...
how TAS is used to make the semaphore operations indivisible. (This lock-and-unlock action is also shown in Figures Four and Five.)

Note that the “busy bit” in the semaphore only means that the semaphore is busy, not that the resource is busy. So that the CPUs don’t spend time waiting on this bit, we ensure that only the routines which set this bit also clear it after a few dozen instructions at most.

Managing Interrupts with WAIT and SIGNAL

We wish to be able to start a task on the occurrence of an interrupt. Presumably, this task will have been in an idle state, waiting for the interrupt. WAIT and SIGNAL let us do this. A semaphore (and queue) are defined for an interrupt, with the difference that the semaphore is initialized to zero instead of +1. The interrupt service task then waits on this semaphore, causing it to be parked on the wait queue.

Some other task will be running when the interrupt occurs. The interrupt handler saves all of the machine context on that task’s stack, then calls (SIGNAL). (SIGNAL) stacks the PC, puts the running task on the ready queue, and starts the service task which was waiting on the semaphore queue.

When the service task completes, it will wait again. Eventually, the task that was interrupted will be pulled from the ready queue and its PC popped from its stack. The PC that was stacked points into the interrupt handler code, just after the call to (SIGNAL). This will be the code to restore the full context and return from interrupt. (Figure Seven shows how the context is stored by WAIT and SIGNAL.)

The task is always resumed at the point in the machine code where it was suspended. This allows a different context to be saved for programmer and interrupt-driven task switches. A high-level task switch (e.g., PAUSE) need only save IP, RP, and SP.

The Listing

Listing One is the 68000 assembler code for the tasker. It was written in a FORTH derivative, so there are some differences from the Forth-83 Standard.

SUBROUTINE defines a code word which simply returns its address when executed. The notation used here is (xxx)
Figure One. The hardware.

Figure Two. A queue.

Figure Three. System queues.

Figure Four. WAIT.

Figure Five. SIGNAL.

Figure Six. Multiprocessor protection.

Figure Seven. A task’s user area.
for an assembly language subroutine; xxx for the executable Forth word.

Screen 181 defines the layout of the task area (Figure Seven). TASK allocates this space in a named data structure.

Screen 182 defines some of the layout of a semaphore queue header. SEMAPHORE is the defining word. Note that the ready queue is defined the same as a semaphore queue.

Every task includes a “private” queue header in its user variables area. This “self queue” is used by SUSPEND and RESUME (described below).

(START) activates a task for the first time by putting it on the tail of the ready queue (see screen 183). From this moment on, except when executing, the task will always be on some queue or other. The remaining words simply move tasks from queue to queue.

(SIGNAL) and (WAIT) are the basic task-control routines. They are callable from a machine-language routine, such as an interrupt handler.

The Forth-callable START, SIGNAL, and WAIT are on screen 189.

Note that a PAUSE (voluntary task switch) is achieved by simply SIGNALing the ready queue. (Follow the logic in Figure Five.)

A task is SUSPENDED by causing it to WAIT on its self queue, whose semaphore is initialized to zero. Another task can SIGNAL that semaphore to RESUME the suspended task.

NEWDEVICE and NEWRESOURCE initialize a semaphore queue for an interrupt and a shared resource, respectively.

INIT-TASK initializes a task area created by TASK. It stacks a context such that the task will begin high-level execution at init-ip (which should be the parameter field address of a colon definition).

Screen 192 illustrates the creation of a task, a do-nothing task in this case. Defining one such task per CPU will ensure that the ready queue is never empty (an error condition).

COLDSTART shows how the multi-tasker, boot task, and defined tasks are initialized in a colon definition. Some such word will be required in the final system’s startup code.

Where to Go From Here

This implementation was adequate for our needs, but it can certainly be taken

(Screens from page 13)

Screen # 11
(MAIN2, MAIN3 SCREEN 21:20 04/29/88)
(Main2 CR
F1 @ DUP ?PRINTER IF
CR ." Vector to be rotated:" CR
"( VX @ . BS ." ; " VY @ . BS ." ; " VY @ . BS :" )" CR IF THEN O ?PRINTER ;

(MAIN3 CR
CR ." Enter components of Quaternion Axis:" CR
". X component:" INPUT VGX ! CR
VGX @ QZ IF
". Y component:" INPUT CR DUP VGX ! QZ IF
". Z component:" INPUT CR VGZ !
ELSE O VGZ ! THEN
ELSE O DUP VGY ! VGZ ! THEN :)

Screen # 12
(MAIN4 SCREEN 21:20 04/29/88)
(Main4 CR
" Angle to rotate (in degrees):" INPUT AGG!
F1 @ DUP ?PRINTER IF
CR ." Quaternion rotation vector:" CR
". X component:" VGX @ . CR
". Y component:" VGY @ . CR
". Z component:" VGZ @ . CR
" Quaternion Gyration Axis: (" VGX @ . BS ." ; " VGY @ . BS ." ; " VGZ @ . BS ." )" CR
" Angle to rotate (in degrees):"
CR IF THEN O ?PRINTER
VGX @ SQR VGY @ SQR + VGZ @ SQR + 0 2 SQR T VGXN
VGX @ MVG @ / VGYN
VGZ @ MVG @ / VGZN ;

Screen # 13
(MAIN5 SCREEN 21:20 04/29/88)
(Main5 CR
AGG @ 2 / DUP DUP DUP
COS 1000 / QO2 ;
SIN 1000 / VGXN @ * Q12 ;
SIN 1000 / VGYN @ * Q22 ;
SIN 1000 / VGZN @ * Q32 ;

(Transfer of values to those in PQ word)
QO2 @ Q12 @ Q22 @ Q32 @
QO1 @ Q11 @ Q21 @ Q31 @
A3 ! A2 ! A1 ! A0 !
E3 ! E2 ! E1 ! E0 !
PQ
QO @ Q1 @ Q2 @ Q3 @
QGT ; QGT ; QGT ; QGT ;
QO7 @ QO1 @ Q11 @ Q21 @ Q2T @ Q21 @ Q3T @ Q31 ! ;

Screen # 14
(CONVERT-TO-NORMAL utilities 21:20 04/29/88)
(For coping with the inexistence of 8087 co-processor... )
(CONVERT-TO-NORMAL utilities
QGT @ Q1T @ Q2T @ Q3T @
R1 @ R2 @ R3 @
7 0 DO
6 ROLL 10000 / LOOP
B3 ! R2 ! R1 !
Q3T ! Q2T ! Q1T ! QGT ! ;
@ /E4 ITER @ 0 DO 10000 / LOOP ;
@ /E2 ITER @ 0 DO 100 / LOOP ;
DIVIDE
R3 @ /E4 R3 ! R2 @ /E4 R2 ! R1 @ /E4 R1 @ QGT @ /E2 QOT @ Q1T @ /E2 Q1T !
Q2T @ /E2 Q2T ! Q3T @ /E2 Q3T ! ;

(Screens continued on next page)
Want more than one turn for the same vector? (1:Y, 0:N): 

INPUT NEGATE;

INPUT NEGATE
Q3T @ NEGATE
A3 ! A2 ! A1 ! A0 !
E3 ! E2 ! E1 ! E0 !
PQ @ Q1 @ Q2 @ Q3 @
D ! C ! B ! A !

QOT @ Q1T @ Q2T @ Q3T @ A @ B @ C @ D @

INPUT NEGATE ;

Total rotation by Quaternions is:
Q= (QOT @ BS) e0+(Q1T @ BS) e1+(Q2T @ BS) e2+(Q3T @ BS) e3 INFORM1

The resultant rotated vector is:
R= (R1 @ BS , R2 @ BS , R3 @ BS) INFORM2 ;

Enter 1 to continue, 0 to stop: 

INPUT NEGATE ;

Total rotation by Quaternions is:
Q= (QOT @ BS) e0+(Q1T @ BS) e1+(Q2T @ BS) e2+(Q3T @ BS) e3 INFORM1

The resultant rotated vector is:
R= (R1 @ BS , R2 @ BS , R3 @ BS) INFORM2 ;

Enter 1 to continue, 0 to stop: 

INPUT NEGATE ;

Support could be included for private tasks, i.e., tasks restricted to one CPU and to that CPU’s memory. This would largely solve the problem of bus saturation.

We have prototyped a round-robin tasker with multiprocessor support; this may be better suited to many applications.

Finally, the principles of the 68000 multiprocessor tasker can be applied to other CPUs!

References

Bradford J. Rodriguez is a freelance software/hardware designer specializing in real-time control applications. He discovered Forth as a student in 1978, but only recently was seduced into speaking and writing about it.
I have developed a simple vocabulary search-order routine in which
VOC1 SEARCHES
VOC2 SEARCHES
VOC3
establishes the specified search order, with
VOC3 and all other vocabularies followed
immediately by FORTH. Entering
NORMAL SEARCH restores the default
condition of each vocabulary being followed
by FORTH in the search order.

My routine is based on a vocabulary
name structure using the dummy link as a
pointer to the last word in the vocabulary,
and a dummy parameter as a pointer to the
dummy link of the next vocabulary to be
searched. (See Figure One.)

A departure from tradition.

CONTEXT/CURRENT point to the
appropriate dummy link, which in turn
points to the last word in <vname>. The
first word in <vname> has its link point to
the dummy name (81A0) in <vname>.
Thus, in an empty vocabulary, the “last
word” is the dummy name of that vocabu-
lar y. See Figure Two for <vname2>. This
is a departure from tradition. The dummy
parameter in <vname> points to the
dummy link of the next vocabulary in the
search order, normally FORTH.

Words needed for constructing and
searching vocabularies: A five-parameter
variable VOC.LIST holding the names of
all vocabularies; and a system variable
SEARCH.VOC playing the role of CON-
TEXT, pointing to the top-word pointer of
the vocabulary to be searched.

(Figures, continued on page 30)
THE CHALLENGE OF SORTS

The Forth Interest Group (FIG) is pleased to announce a challenge to all Forth programmers. Beat our sort program and have a chance to win a prize of your choice. The author of the program judged best in our tests will get to choose between free online access to the FIG RoundTable on GEnie for one month, a $150 credit toward purchases from the FIG Mail Order Form, or a check for $100.

The Rules
Submissions must be electronically transmitted to the Software Libraries in the FIG RoundTable on GEnie no later than midnight November 31, 1989. The results and the winning entry will be published in the March/April issue of Forth Dimensions. All entries and results will be available on most Forth Bulletin Board systems soon after testing is complete. All submissions become the property of the Forth Interest Group for distribution as it sees fit. The source code for all entries must comply with the Forth-83 Standard (published in 1983 by the Forth Standards Team), a document available on the FIG Mail Order Form. The source code may be submitted in text or block format, but must comply to the conventions in the block file SORT.BLK (see following). Submissions will be compiled and tested with this test suite, and the average score after 80 TESTS will be used to compare it to other submissions. The examples included in SORT.BLK provide best- and worst-case examples for sorting algorithms. The BUBBLE sort is the simplest, and the QUICK sort is a modification (by Wil Baden) of a sorting algorithm developed by C. Hoare. Figure One gives a sample of the statistics generated by each on the judges’ system.

Although we encourage you to beat the score of our QUICK sort, that is not necessary to win this competition. The winner will be chosen from the valid submissions, based on the lowest average score (the last entry in the right-hand column after 80 TESTS). Submissions will be disqualified if they do not comply to the Forth-83 Standard or if they fail to execute under this test suite.

Test Details
Dictionary bytes are determined by the size of the submitted sort after being compiled into our version of Forth. This Forth is based on the popular F83 model developed by Harry Laxen and Mike Perry. Although it is upwardly compatible with F83, we do not guarantee that the entire test suite will run under your version. In addition, since each version of Forth differs in how it compiles source code, do not assume that you can duplicate our results.

Figure Two shows statistics about how our Forth compiles source code. It is not intended to be a complete list of how our version of F83 works, but should give you an indication of how it differs from the F83 model. Refer to Inside F83 by C.H. Ting (see the FIG Mail Order Form) for more complete details.

RAM words are determined by memory usage outside the Forth dictionary; this includes the parameter and return stacks, PAD, TIB, and any other memory usage between these areas and the top of the dictionary (referenced by HERE). See the table cited above for indications of how this number is affected; I have found it extremely hard to calculate, and have noticed that it is high by about 26 items. However, it is sufficient for the sake of this test.

Fetches and stores are affected by access to the DATA array to be sorted. They are incremented by the words S@ and S! which must be used for all accesses into the DATA array.

Although this test suite is only based on a sort of 1024 bytes, it would not be useful to limit a sorting algorithm to this size. It should be assumed that there could be an unlimited number of data items and that the data could be of any size. The sorting algorithm should be easily modifiable to accommodate any variations in the data format.

Comparer are incremented by use of the word COMPARE which also must be used in your sorting algorithm. It will return a number that represents the difference between two data items, according to the following truth table:

\[
\begin{array}{c c c}
 n_1 & n_2 & \text{result} \\
 1 & 2 & 1 \\
 1 & 1 & 0 \\
 2 & 1 & -1 \\
 \end{array}
\]

The execution time is based the MS-DOS time function call which returns the current time down to 1/100th of a second. Although it is generally accurate, it has shown variations of up to ±5/100ths of a second. This should not be significant, though, since the time is scaled by the number of bytes we are sorting and will only give us an error of ±5/102400ths in our final score. As best as possible, we have tried to isolate the execution time of the sort itself, but there is a slight overhead encountered that is not measurable on our test machine. The tests will be run on a 12.5 Mhz 80386 computer running MS-DOS version 3.21. The score is based on a calculation combining all the other numbers in the following formula:

\[
\frac{((\text{Fetches}+\text{Stores}+\text{Compares})+((\text{Dict}+\text{RAM}) \times \text{Time})/100)}{\text{BYTES}}
\]

This will weight the memory usage
based on the amount of time the sort takes to execute, and will scale everything by the number of bytes being sorted. Although this is a fairly arbitrary measure of efficiency, it makes a sort that minimizes data access come out with the lower score. Under normal conditions, this could be considered the goal of any sorting algorithm.

The maximum is the score based on the individual maximums of each of the above items. This will indicate a worst case for the sorting algorithm. However, it is highly unlikely that the results would ever be produced on any one test. This number will only be used to resolve a tie.

The average is the score based on the individual average of each of the above items. It should indicate how the sort will perform under a variety of situations. This is the number we will use as the basis of our comparison.

### The Data

The DATA array contains 1024 ITEMS to be treated as 16-bit signed values.

There are eight types of data patterns that we will cycle through during the tests. Each pattern will be used ten times during our test, and each will contribute to the scores:

- The RAMP is a simple array of ascending values. This array is already sorted, so it should produce the lowest score.
- The SLOPE is also a simple array, but of descending values. The values in the array need to be reversed.
- The WILD pattern contains random signed values in each element.
- The SHUFFLE pattern starts with the RAMP, then reorders each of the elements into a random pattern.
- The BYTE pattern consists of random eight-bit values. There will obviously be some duplication in this array.
- The FLAT array is filled with a single value. It will be a random value, but the array does not need to be rearranged.
- The CHECKER pattern consists of alternating values. Two random values are selected and placed into the even and odd addresses.
- The HUMP is a Gaussian distribution of values. This pattern has a bell shape when viewed in graphic format.

### The Analysis

As described earlier, we selected a scoring system based on the criteria we consider important in a sorting algorithm. However, do not expect that you will be able to reproduce our exact results. To make timing comparisons before you submit your entry, base them on the results you generated by the judges' quick-sort and bubble-sort routines.

#### QUICK SORT

<table>
<thead>
<tr>
<th>Test</th>
<th>Dict</th>
<th>RAM</th>
<th>Fetched</th>
<th>Stores</th>
<th>Compares</th>
<th>Time</th>
<th>Score</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMP</td>
<td>400</td>
<td>50</td>
<td>9348</td>
<td>1023</td>
<td>7944</td>
<td>2.03</td>
<td>18.77</td>
<td>18.77</td>
<td>18.77</td>
</tr>
<tr>
<td>SLOPE</td>
<td>400</td>
<td>51</td>
<td>10383</td>
<td>2050</td>
<td>7951</td>
<td>2.26</td>
<td>20.89</td>
<td>20.89</td>
<td>19.83</td>
</tr>
<tr>
<td>WILD</td>
<td>400</td>
<td>46</td>
<td>17793</td>
<td>5881</td>
<td>11228</td>
<td>3.79</td>
<td>35.73</td>
<td>35.74</td>
<td>25.12</td>
</tr>
<tr>
<td>SHUFFLE</td>
<td>400</td>
<td>52</td>
<td>17823</td>
<td>11253</td>
<td>3.79</td>
<td>35.81</td>
<td>35.81</td>
<td>27.80</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>400</td>
<td>46</td>
<td>16317</td>
<td>10201</td>
<td>3.46</td>
<td>32.76</td>
<td>35.81</td>
<td>28.79</td>
<td></td>
</tr>
<tr>
<td>FLAT</td>
<td>400</td>
<td>52</td>
<td>16255</td>
<td>7810</td>
<td>3.51</td>
<td>32.91</td>
<td>37.69</td>
<td>29.47</td>
<td></td>
</tr>
<tr>
<td>CHECKER</td>
<td>400</td>
<td>51</td>
<td>16668</td>
<td>8595</td>
<td>3.63</td>
<td>33.61</td>
<td>37.69</td>
<td>30.06</td>
<td></td>
</tr>
<tr>
<td>HUMP</td>
<td>400</td>
<td>43</td>
<td>15858</td>
<td>5575</td>
<td>3.40</td>
<td>31.87</td>
<td>37.69</td>
<td>30.28</td>
<td></td>
</tr>
</tbody>
</table>

#### BUBBLE SORT

<table>
<thead>
<tr>
<th>Test</th>
<th>Dict</th>
<th>RAM</th>
<th>Fetched</th>
<th>Stores</th>
<th>Compares</th>
<th>Time</th>
<th>Score</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMP</td>
<td>52</td>
<td>40</td>
<td>1047552</td>
<td>1023</td>
<td>7944</td>
<td>2.03</td>
<td>18.77</td>
<td>18.77</td>
<td>18.77</td>
</tr>
<tr>
<td>SLOPE</td>
<td>52</td>
<td>43</td>
<td>2095104</td>
<td>1047552</td>
<td>523776</td>
<td>0</td>
<td>523776</td>
<td>1548.45</td>
<td>1548.45</td>
</tr>
<tr>
<td>WILD</td>
<td>52</td>
<td>43</td>
<td>1552494</td>
<td>504942</td>
<td>523776</td>
<td>250.57</td>
<td>2543.95</td>
<td>3613.22</td>
<td>2580.57</td>
</tr>
<tr>
<td>SHUFFLE</td>
<td>52</td>
<td>43</td>
<td>1542996</td>
<td>495444</td>
<td>523776</td>
<td>248.75</td>
<td>2525.23</td>
<td>3613.22</td>
<td>2588.44</td>
</tr>
<tr>
<td>BYTE</td>
<td>52</td>
<td>43</td>
<td>1240412</td>
<td>192860</td>
<td>523776</td>
<td>191.85</td>
<td>1928.96</td>
<td>3613.22</td>
<td>2557.58</td>
</tr>
<tr>
<td>FLAT</td>
<td>52</td>
<td>43</td>
<td>1047552</td>
<td>0</td>
<td>523776</td>
<td>155.77</td>
<td>1548.49</td>
<td>3613.22</td>
<td>2431.82</td>
</tr>
<tr>
<td>CHECKER</td>
<td>52</td>
<td>40</td>
<td>1048574</td>
<td>1022</td>
<td>523776</td>
<td>155.88</td>
<td>1550.49</td>
<td>3613.22</td>
<td>2284.65</td>
</tr>
<tr>
<td>HUMP</td>
<td>52</td>
<td>43</td>
<td>1163672</td>
<td>116120</td>
<td>523776</td>
<td>177.46</td>
<td>1777.75</td>
<td>3613.22</td>
<td>2179.81</td>
</tr>
</tbody>
</table>

**Figure One.** Sample statistics generated by the judges' quick-sort and bubble-sort routines.
obtain from running our examples on your computer. We will run the test 80 times, cycling through each data pattern ten times. We will upload the results from the last eight runs of each submission into the Bulletin Board section of the Forth RoundTable on GEnie, showing the individual scores for each data pattern. From there, they will be distributed to the other Forth Bulletin Board systems within our virtual network. The score based on the averages after the last run will be used to rank each entry. If there is a tie between two entries, we will use the score based on the maximums to break the tie. If there is still a tie, we will select the winner based on the readability of the source code and the documentation included with it. We will publish the three entries with the lowest scores in the March/April issue of Forth Dimensions.

All entries must either be uploaded to the Software Libraries of the Forth RoundTable on GEnie or mailed to the FIG business offices (P.O. Box 8231, San Jose, California 95155 U.S.A.), where they will be uploaded for you. All entries must contain the name, address, and telephone number of the author so that winners can be notified. The deadline for submissions is November 30, 1989. All submissions become the property of the Forth Interest Group.

May the best sort win!

---

**Figure Two. Examples of how the judges' Forth compiles source code.**

<table>
<thead>
<tr>
<th>Construct</th>
<th>Dictionary</th>
<th>RAM use</th>
</tr>
</thead>
<tbody>
<tr>
<td>: Header</td>
<td>4 bytes</td>
<td>1 word</td>
</tr>
<tr>
<td>DO</td>
<td>4 bytes</td>
<td>6 words</td>
</tr>
<tr>
<td>LOOP</td>
<td>4 bytes</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>4 bytes</td>
<td></td>
</tr>
<tr>
<td>ELSE</td>
<td>4 bytes</td>
<td></td>
</tr>
<tr>
<td>UNTIL</td>
<td>4 bytes</td>
<td></td>
</tr>
<tr>
<td>; etc.</td>
<td>2 bytes</td>
<td></td>
</tr>
<tr>
<td>16-bit literal</td>
<td>4 bytes</td>
<td>1 word</td>
</tr>
<tr>
<td>32-bit literal</td>
<td>6 bytes</td>
<td>2 words</td>
</tr>
</tbody>
</table>

---

**SORT.BLK**

[Also available for downloading from the GEnie Forth RoundTable.]

```
Screen 0
0 \ SORT.BLK A Sorting Competition 11Jun89dar
1 2 ( Consider this a challenge. The Forth Interest Group wants)
3 ( to see how good you are. Come up with a sort that will beat)
4 ( this one and win your choice of valuable prizes. Read the)
5 ( documentation file that accompanies this source for complete)
6 ( details about the prizes and rules for participation)
7 ( May the best sort win. DaR )
8 9 ( As in all my code since 1986 the stack at the beginning of )
10 ( a line not starting with a control flow word, and at extra )
11 ( space in the middle of a line, is given by the most recent )
12 ( stack comment. For Forth to be readable it is absolutely )
13 ( necessary that what is on the stack is known. After a)  
14 ( control flow word the stack is given by extra space or a )
15 ( stack comment. WWB )
```
Screen 2

0 \ Data Array and Utilities 11Jun89dar
1 \ CELLS ( a -- a': P: Scale word size ) 2* ;
2 \ CELLS ( a -- a': P: Scale double size ) 2* 2* ;
3 # 1024 CONSTANT ITEMS ( -- # P: Number of data items to sort )
4 CREATE DATA ( -- a P: Data to be sorted ) ITEMS CELLS ALLOT
6 # 7: D* ( dn dm -- dp P: Double number multiply )
8 \ >R SWAP OVER ( n1 n2 n1 m1 ) \ >R ( n1 m1)
9 \ OVER \ VMU* ( dp ) \ >R \ >R SWAP \ >R
10 ( dp n2*n1n2 m2 ) * + + ;
11 # 12: MU/NEAR (dn nd -- dq P: Double divide with rounding)
13 DUP \ >R MU/MOD ( r dq ) \ >R \ >R ( r )
14 2* \ R@ 1 AND + \ >R \ >R ROT ( dq ) R
15 \ >IF ( dq ) 1 M+ THEN ;

Screen 3

0 \ Data Access Statistics 11Jun89dar
1 # VARIABLE TIMES ( -- a P: Number of tests we have completed )
2 # 3: \ USE ( a -- P: Increment usage counter )
4 DUP 2@ ( a d ) 1, D+ \ >R OT 2! ;
5 # 6: \ MAX ( a -- P: Store unsigned maximum ) DUP \ >R 2@ ( d )
7 \ R@ 1 CELLS + 28 ( d0 d1 ) 2OVER 2OVER \ DU<
8 \ IF 2SWAP THEN 2DROP ( d ) \ >R 1 CELLS + 2! ;
9 # 10: \ AVG ( a -- P: Accumulate average ) DUP \ >R 2@ ( d )
11 \ R@ 2 CELLS + 28 ( d0 d2 ) TIMES @ S>D \ D*
12 \ D+ ( d ) TIMES @ 1+ MU/NEAR \ >R 2 CELLS + 2! ;
13 # 14: \ RESULTS ( a -- P: Analyze ) DUP \ >AVG \ MAX ;
15 #

Screen 4

0 \ Data Access Utilities 11Jun89dar
1 # 2VARIABLE FETCHES ( -- a P: Times fetched ) 2 CELLS ALLOT
2 2VARIABLE STORES ( -- a P: Times stored ) 2 CELLS ALLOT
3 # 2VARIABLE COMPARES ( -- a P: Times compared ) 2 CELLS ALLOT
4 # 5: \ S? ( # -- n P: Must be used to fetch value )
6 CELLS DATA + @ ( n ) FETCHES \ USE ;
7 # 8: \ S! ( n # -- P: Must be used to store value )
9 CELLS DATA + ! ( n ) STORES \ USE ;
10 #
11 # COMPAR ( n1 n2 -1| 0| 1 P: Must be used for compares )
12 \ 2DROP \ >R 1 \ AND ( t ) \ >R OR \ COMPARES \ USE ;
13 #
15 #

Screen 5

0 \ Bubble Sort Example 11Jun89dar
1 : EXCHANGE ( #1 #2 -- P: Exchange items at indices )
2 \ 2DROP \ >R SWAP \ >R OT \ SWAP \ >R
3 # 4: \ BUBBLE ( # -- P: Slow sort for comparison )
5 1 DO 0 DO J \ >S I \ >S \ COMPARE
6 \ 0< IF I \ >J EXCHANGE THEN
7 \ LOOP \ LOOP ;
8 # 9: \ TIMER ( d1/100s -- P: Display timer in seconds )
10 \ 1\ # # 46 (. ) HOLD \ RS #
11 \ ( a # ) \ 8 \ OVER \ - \ SPACES \ TYPE ;
12 #
13 #
14

Screen 6

0 \ Quick Sort Utilities 11Jun89dar
1 : ORDER-3
2 ( f l - f l # P: Order first, middle and last index )
3 \ 2DROP \ >R 0 32767 AND \ >R
4 \ DUP \ >R \ >R \ COMPARE 0< IF \ DUP \ >R EXCHANGE THEN
5 \ OVER \ >R \ >R \ COMPARE 0> IF \ OVER \ >R EXCHANGE THEN
6 \ THEN \ >R ;
7 # 8: \ BOTH-ENDS ( f l p -- f l' P: Trim ends ) \ >R ( f l )
9 \ BEGIN OVER \ >R \ >R \ COMPARE 0< WHILE 1 0 \ >D \ REPEAT
10 \ BEGIN DUP \ >R \ >R \ COMPARE 0> WHILE 1- \ REPEAT
11 \ \ >R \ DROP ;
12 #
13 #
14

TDS 9090 FORTH COMPUTER
Ideal for starter, teaching or target system

- build into your product for rapid completion!
- program with IBM-PC

- complete Fig-Forth system
- connect to keyboard, Icd display, RS 232
- 30K RAM, 16K EPROM
- 35 I/O lines; 10 bit ADC option
- over 3000 in use in Europe
- low power - down to 3 ma @ 6-16v

Connect the 4" x 3" TDS 9090 single-board computer to an IBM-PC or compatible and start writing Fig-Forth code immediately! Lots of ready made application programs come with the kit to do interrupt-driven I/O, graphics, hobbies, frequency measurement, solid-state speech and data-logging. The board includes a ROM-resident Forth language kernel and an assembler. By storing generated code in either non-volatile RAM or EPROM, the board can be used in a target system or stand-alone product. Based on the CMOS Hitachi HD 63A03Y microprocessor, it has two timers, two serial ports and interrupts which are available via Forth instructions. Also included on board are 30K RAM for storing source code or data, 16K EPROM/16K for firmware, 256 bytes EEPROM, 35 I/O lines, two RS 232 serial interfaces, a watchdog timer to insirre recovery from crashes, and an expansion bus. The circuit board 9090 to an 8 x 8 keyboard or an Icd display, or use two of the I/O lines as an IEC interface. The board is a resident Forth is an extended version of Fig-Forth with Forth words to support all the onboard peripherals, as well as the keyboard and Icd interface. Put product application software into PROM and it starts to run as soon as power is applied. Made in England by Triangle Digital Services, and well-known in Europe, the TDS 9090 is now supported in the USA and is available with less than two-week delivery at only

$219 (25qty)

The S2lq Company 1183 Moseley Rd Victor NY 14564 USA
tel: (716) 425-4367 or fax (716) 425-7391
Screen 7
0 \ Quick Sort List Processing 11Jun89dar
1 \PARTITION ( f l -- f l' f l' P: Rearrange lists )
2 ORDER-3 S$ >R 2DUP l-1 D+ ( f l f l' l' )
3 BEGIN R$ BOTH-ENDS 2DUP l+1 U<
4 IF 2DUP EXCHANGE l-1 D+ 1
5 THEN 2DUP SWAP U<
6 UNTIL R> DROP SWAP RO ;
7 \SINK ( f p # -- f P: Do insertion ) ROT >R ( p # )
9 BEGIN 1- 2DUP S$=COMPARE 0c
10 WHILE DUP S$ OVER l+ S$ DUP R$ =
11 IF S! ( ) R> EXIT THEN
12 \REPEAT 1+ S! ( ) R> ;
13
14
15

Screen 8
0 \ Quick Sort Algorithm 11Jun89dar
1 \INSERTION ( f l -- P: Insertion sort ) 2DUP U<
2 IF l+ OVER l+ DO ( f ) l S$ I SINK LOOP DROP
3 ELSE 2DROP THEN ;
4 \HOARIFY ( f l -- ...P: Quick and Insertion sorts )
6 BEGIN 2DUP 7 0 D+ U< WHILE PARTITION ( f l f' l' )
7 2DUP -R 2OVER -R> IF 2SWAP THEN
9 \REPEAT 2DROP ;
10 \QUICK ( # -- P:Quick sort ) 1- 0 SWAP DEPTH >R
11 BEGIN ( ... ) HOARIFY DEPTH R$ < UNTIL R> DROP ;
12 \ QUICK IS SORT
13
14 \SINKING ( n-- P:Insertion Sort)1-0 SWAP INSERTION;

Screen 9
0 \ Random Number Generator 11Jun89dar
1 \VARIABLE SEED ( -- a P: Random data pattern )
2 3 : SETUP ( -- P:Setup random sequence) 1234 SEED ! ;
5 \RANDOM ( -- n P: Calculate next random number )
6 SEED # ( n ) 314159261 * 1+ DUP SEED ! ;
7 \CHOOSE
8 ( limit -- 0..limit-1 P:Choose next random in range)
9 \ RANDOM ( limit n ) UM* SWAP DROP ;
10 \GAUSS ( n-- u P: Gaussian distribution )
11 \ RANDOM 0 \ RANDOM D- RANDOM 0 D+ 1
13 \ RANDOM 0 D+ \ RANDOM 0 D+ \ RANDOM 0 D+
14 \ UM/MOD SWAP DROP UM* SWAP DROP ;
15

Screen 10
0 \ Random Data patterns 11Jun89dar
1 \ RAMP (-- P: Ascending values) ITEMS 0DO I I S! LOOP ;
2 \ SLOPE ( -- P: Build sample of descending values )
3 ITEMS 0 DO ITEMS 1- I S! LOOP ;
4 \ WILD ( -- P: Build sample of random positive values)
5 ITEMS 0 DO RANDOM I S! LOOP ;
6 \ SHUFFLE ( -- P: Build sample of shuffled sequence) RAMP
7 ITEMS 0 DO ITEMS CHOOSE I EXCHANGE LOOP ;
8 \ BYTE ( -- P: Build sample of byte values )
9 ITEMS 0 DO 256 CHOOSE I S! LOOP ;
10 \ FLAT ( -- P: Build sample of equal values) RANDOM ( n )
11 ITEMS 0 DO DUP I S! LOOP DROP ;
12 \ CHECKER ( -- P: Checker board) RANDOM RANDOM ( n1 n2)
13 ITEMS 0 DO DUP I S! SWAP LOOP 2DROP ;
14 \ HUMP ( -- P: Gaussian or bell curved data )
15 ITEMS 0 DO 256 GAUSS I S! LOOP ;

Screen 11
0 \ Pattern Setup and Analysis 11Jun89dar
1 \ PATTERNS ( -- P: Group data setup patterns )
2 \ RAMP SLOPE WILD SHUFFLE
3 \ BYTE FLAT CHECKER HUMP ;
4 \ PATTERN #--P: Set up data by test) DUP TIMES ! 8 MOD
6 \ CELLS \} PATTERNS BODY + ( cfa ) PERFORM ;
7 \ TEST-DATA(-- P: Check order of data) DATA @ ITEMS1
9 DO ( prev ) DATA I CELLS @ SWAP OVER >
10 \ ABORT" Data has not been sorted"
11 \ LOOP DROP ;
12
13
14

Screen 12
0 \ Stack Usage Checks 11Jun89dar
1 2VARIABLESTACK( -- a P: Sum of RAM usage)2 CELLS ALLOT
2 \ HEX 5A5A5A5A5CONSTANTMARK ( -- n P: Stack mark) DECIMAL
4 \ :FILL-\RAM ( -- P: Fill RAM with Markers) MARK HERE !
6 \ HERE DUP 1 CELLS + \ R8 OVER - CMOVE ;
7 \ :TEST-\RAM ( -- P: Check RAM usage )
9 0 \ STACK2! HERE 1- \ CELLS NEGATE AND ( a )
10 BEGIN DUP @ MARK \ IF STACK USE THEN
11 \ BEGIN DUP @ MARK - IF STACK USE THEN
12 \ STACK !RESULTS ;
13
14
15
Screen 13
0 \ Setup Sort Tests 11Jun89dar
1 2VARIABLE TIME ( -- a P: Sum of time) 2 CELLS ALLOT
2
3 : !TIME ( d1 d2 -- P: Store timing results)
4 2SWAP D- TIME 2! TIME 1RESULTS;
5
6 : TEST-SORT ( -- P: Test the sort algorithm)
7 0. FETCHES 2! 0. STORES 2! 0. COMPARES 2!
8 FILL-RAM COUNTER ( d ) ITEMS SORT
9 COUNTER ( d1 d2 ) TEST-RAM !TIME ( )
10 FETCHES 1RESULTS STORES 1RESULTS
11 COMPARES 1RESULTS;
12
13

Screen 14
0 \ Sort Test Reports 11Jun89dar
1 : HEADER ( -- P: Set up and display test header)
2 FETCHES 3 CELLS ERASE STACK 3 CELLS ERASE
3 STORES 3 CELLS ERASE TIME 3 CELLS ERASE
4 COMPARES 3 CELLS ERASE SETUP CR
5 "." Test Dict RAM Fetches Stores Compares ".
6 " Time Score Maximum Average";
7
8 : RESULTS ( n -- P: Display results)
9 >R ( ) DICTIONARY 4 U.R
10 STACK R@ 2 CELLS + 8 4 U.D.R
11 FETCHES R@ 2 CELLS + 8 8 U.D.R
12 STORES R@ 2 CELLS + 8 8 U.D.R
13 COMPARES R@ 2 CELLS + 8 8 U.D.R
14 TIME R> 2 CELLS + 8 .TIMER ;
15

Screen 15
0 \ Report test results 11Jun89dar
1 : .ANALYSIS ( n -- P: Calculate results) >R ()
2 FETCHES R@ 2 CELLS + 20 ( dfetch )
3 STORES R@ 2 CELLS + 20 ( dfetch dstore )
4 COMPARES R@ 2 CELLS + 20 ( dfetch dstore dcomp )
5 D+ D+ 100 ITEMS M*/ ( df+d+dc/items )
6 TIME R@ 2 CELLS + 20 ( d dtime )
7 STACK R> 2 CELLS + 20 DROP ( d dtime stack)
8 DICTIONARY + ITEMS M*/ ( d1 d2) D+.TIMER ;
9
10 : RESULTS ( -- P: Display test results) CR TIMES @ ( n)
11 8 MOD CELLS [ ALSO BUG ] ' PATTERNS >BODY +
12 ( cfa ) >NAME & L.10 ( ) 0 .RESULTS
13 3 0 DO I .ANALYSIS LOOP [ PREVIOUS ];
14
15

Screen 16
0 \ Random generator tests 11Jun89dar
1 VARIABLECYCLE ( -- a P: Random cycle check) 4 CYCLE !
2
3 : TALLY ( n -- P: Show n) BASE ®SWAP [ [base] n ]
4 36 BASE ! 1.R ( [base] ) BASE !;
5
6 : TEST-RANDOM ( -- P: Test generator)
7 PAGE DATA ITEMS CELLS ERASE ITEMS ( k ) 1 1
8 DO ITEMS CHOOSE ( k u ) DUP 64/MOD AT CELLS DATA +
9 ( k a ) DUP >R @ ( k @tally ) DUP 0=
10 IF SWAP 1- SWAP THEN
11 1+ DUP TALLY R>! ( k ) DUP 0=
12 IF 0 18 AT I U. LEAVE THEN
13 I CYCLE @ MOD 0= IF PAGE THEN
14 LOOP DROP ;
15

Indelko RTX Forth Kit
Experimental Kit for Harris RTX2000 Forth Chip including:
One 100x100 mm square PC circuit board
Two EPROM's containing cmForth for RTX
cmForth source code on MS-DOS diskette
Assembly instructions and documentation
$150.00

'More on NC4000' RTX Special Issues:
Volume 10: RTX cmForth and papers
Volume 11: RTX Supplement to 'Footsteps' and SC32 paper and documentation.
$15.00 per volume

Fat Forth for IBM PC/XT/AT: F-PC 2.25
Disk set: four 360K diskettes, $25.00
F-PC User's Manual, $20.00
F-PC Technical Reference Manual, $30.00

Offete Enterprises, Inc.
1306 South B Street
San Mateo, CA 94402
(413) 574-8250

Volume XI, Number 3

Forth Dimensions
phrases they lead to. Personally, I can stand writing TIME > instead of DETECTED, etc., and it keeps the number of required words to a minimum—I mention them for purposes of discussion.

Choice of Time Units

Another point of ongoing discussion is the design/choice of the set of time units. Some people argue that milliseconds are all that is ever required (the code can easily be simplified to this end, if desired), but others feel that a multiplicity of time units is more complete and leads to more readable code.

My own feeling on this issue is that a variable for millisecond-of-minute (0–59,999 unsigned) and another variable for minute-of-week (or even minute-of-month) provides millisecond resolution over a period of more than 45 days in a standard Forth double number.

Also, in this design NEW, LAPSE, etc. could be run as infrequently as every 59.99 seconds and still provide exact millisecond calculation of elapsed times. In the end, this decision depends on the source of the timing information—if there is a real-time clock in the system, I usually implement whatever the hardware provides.

Dave Edwards is a qualified electronic engineer who formed Jarrah Computers—a microprocessor engineering consultancy using Forth as a key element—four years ago. His company has specialized in the design of custom microcontrollers, ranging from the 68705 single-chip family to large industrial systems based on Rockwell’s 65F11 Forth chip and, recently, Motorola’s 68HC11.

(Figures’ figures, from page 23)

<table>
<thead>
<tr>
<th>vname</th>
<th>link</th>
<th>code</th>
<th>set.context</th>
<th>dummy name</th>
<th>dummy link</th>
<th>dummy parm</th>
</tr>
</thead>
</table>

Figure One. A dummy link and parameter make the vocabulary connections.

![Diagram of dummy link and parameter connections]

Figure Two.
In my rush to demonstrate how the GENie Forth RoundTable was involved in the standards effort, how erudite and informative the guests in our real-time conference are, and other impressive bits, I overlooked a facet that may be one of our most important services. This is how we stand as a resource center not only to the Forth expert, but also—perhaps even especially—to the new users of Forth.

I must begin with the Sunday night "Figgy Bar," usually conducted by Leonard Morgenstern. Leonard, and sometimes lead sysop Dennis Ruffer, conduct learning and technical sessions aimed at the new and intermediate Forth user. I have never come from these Sunday FIGGY’s without some better understanding of Forth, so do not assume it is only for beginners. The point is, it is especially for beginners. No question is trivial, so the first step to learning Forth the GENie way is the Sunday night real-time conference.

Also, several files in the library can assist the newcomer. Browsing just the keyword “tutorial” generates an impressive list of files worth looking at, including Bill Kibler’s Forth tutorial written in Forth. All one needs to do is load this file in any Forth-83-compatible system (the public-domain version of F83 for your computer is also waiting in the library!), invoke Kibler’s program, and then learn Forth in Forth.

In the bulletin board area, we also have Category 15, Topic 1: Jack Brown’s F-PC Forth tutorial. Jack has created the best online Forth tutorial I have ever seen. It is intended for use with Tom Zimmer’s F-PC, a Forth for PCs and clones. It can be followed using other kernels, though, and a companion text file for F83 is in the library.

What if someone just has a question? There is lots and lots of help available on the GENie Forth RoundTable bulletin board. Answers are quick to come from the GENie sysops and other GENie users, or via ForthNet, which ties us to several other Forth gurus. Topics such as “Which Public-Domain Kernel” (Category 1, Topic 7), “Basics of the Forth Language” (Category 2, Topic 1), and “for us beginners? HELP” (Category 2, Topic 5) are obviously in place to serve the new Forth user.

Some sample problems and responses follow:

Category 1, Topic 7, Message 1
From: Todd Natkin
Subj: F-PC, F83, MMS FORTH, etc.

A simple question: Is F-PC the “correct” implementation of Forth for me to be learning? Is it considered the most current of the public-domain implementations? I have looked over the material downloaded and ordered the technical reference manual from Dr. Ting, but do not have the time to review all the different versions of Forth and then pick the best one.

Where do you stand on this issue?

Category 1, Topic 7, Message 2
From: Jerry Shifrin
Subj: F-PC, F83, MMS FORTH, etc.

A simple question: Is F-PC the “correct” implementation of Forth for me?

Like, politically correct? It’s on the approved list, okay?

You can do useful work with most of the available Forths. F-PC is good in that it has numerous add-ons already available. OTOH, there may be too much material for some people to be comfortable. For a smaller implementation, check out Martin Tracy’s ZenForth.

Category 1, Topic 7, Message 3
From: John Somerville

Subj: F-PC, F83, MMS FORTH, etc.

Todd, hope you don’t mind me butting in, but I am a relative newcomer to Forth and computing, so my experience may be of interest to you. I have tried several PD Forths and came to the following conclusions:

Laxen & Perry’s Forth (F83) is very good, particularly if you pick up Jack Brown’s VEDIT. However, it does not have floating point, nor graphics. You can metacompile it to run on your hard drive.

Zimmer’s Forth is nice, particularly since Jack Brown has put out floating point for it. I have not tried the floating point, since I have purchased a commercial package. However, I found it too large.

I tried ZenForth, but there were a few versions all packed together, and I really had trouble knowing what documentation referred to what.

UniForth has a demo package which didn’t look too bad, but I didn’t feel right about using it and I had no luck getting in touch with the company.

I purchased the commercial package (UR Forth) because I was tired of fishing around without documentation. Also, I started using versions which someone had altered and I frequently found myself lost in an undocumented morass. However, I think all the boards on this network have virgin copies of the F83 and F-PC systems.

If I had to choose a public-domain Forth now, I would go for F-PC because of the floating point; but F83 still has more appeal because of its compactness. If you choose either of these two, I recommend getting a hard copy of the source code and the user manuals available through FIG.

regards
NET/MAIL: British Columbia Forth Board Burnaby, BC 604-434-5886
Category I, Topic 7, Message 4
From: GARY-S [Gary]
MM20, and MX80 are text-oriented
Forth kernels written for CP/M machines,
while F-PC is written for PCs and com-
patibles. I hope this helps resolve some of
the confusion.

Category I, Topic 7, Message 5
From: M.Hawley
I’ve been going Forth for over a year
now. My recommendation is to
load only applications written for
your particular version of Forth. Otherwise you
will go nuts trying to supply the “missing
word” which hangs your loading process.
With F-PC, you will have the luxury of a
screen editor and sequential files to work with.
You will need the documentation
from Dr. Ting. Enjoy!

By the way, when you get stuck on a
problem, don’t be shy. Post a message to
this board and the experts here will pitch in
to help. They helped me several times. Let
us know how you’re doing...
a recent beginner — meh

Topic 33
From: J.Ventola
Sub: neophyte needs F83 examples
This topic is for pointing us neophytes
to examples in F83 of doing simple things
like getting input from a user.

Category I, Topic 33, Message 26
From: K.Smith10
Just going over these messages for the
first time and noticed some questions I
(finally) might help with... One of the
handiest things I found with using F83
(MS-DOS) is that you can load a screen—
a single screen—from another file while
you are in the process of loading screens
(blocks) from a different application... a
good example of this is the
UTILITY.BLK screen 1; each of these
screens is a list of LOAD instructions for the
screens within its file, and all of this can be
redirected or cancelled or added to as
needed. I do my development with an
F83.COM version that has all the utilities I
might need or want; then, when I’ve fin-
ished my application, I take its file and only
load what it needs—usually not a screen
editor or debugger or dumping, etc. The
load screens act as a vector table pointing to
what you want to use, without having to
physically copy a screen into your applica-
tion file (you do that at compile time in
memory).

Which reminds me of something else
that came up in the messages above, which
is that Forth code is pretty portable—I
know, I know, I’ve had some real fun trying it—but most code is gonna follow, or build
on, accepted Forth fundamentals. If the
original programmer was careful, you’ll
find most of the CPU- or system-specific
code factored out from the general code
(i.e., if you’re going to write directly to
screen memory and bypass the standard
system calls, which words like EMIT are
usually built on, that code will be off in its
own screen grouped with supporting code,
all of it building up to provide the whole
application with generalized words like
“print”—you could rewrite the low-level
screen-memory codes to use your system
addresses, etc., or simply make up “print”
from general Forth output words like
EMIT.

Long winded! You’d think I was a For-
tran programmer!

Category I, Topic 33, Message 27
From: K.Smith10
Thought I’d better split up these replies
into separate notes. J. Ventola brought up
implementing Pilot in Forth, but also men-
tioned that he’d found a cheap version
available, so... but I bet some of the useful
qualities of Pilot would be handy, at least as
a module, within Forth. For a reference on
Pilot, I remember an article in Computer
Language magazine, the July and Septem-
ber 1986 issues, titled “Interpreter Design
and Construction, formal language defini-
tion and initial coding in Pilot.” In the
article, the subject really is formal language
definition, but the vehicle is to define Pilot.
Not sure, don’t remember how strictly Pilot
is actually followed, but the article will
provide ideas on how to go about imple-
menting a language, as well as discussing
the attributes of Pilot.

Category I, Topic 33, Message 28
From: K.Smith10
Computer Language magazine has
been a great and enjoyable resource for me
over the years, but for lots of Forth refer-
ence I recommend Dr. Dobb’s Journal.
Martin Tracy’s “4th Column” would be
interesting to a new or old Forth program-
er. Something that has helped me under-
stand and use Forth better is to look at other
languages (for which it is often easier to
find a larger variety of subjects covered
and, generally, more references), and also
to look into more general aspects of com-
puter programming. I’ve found that, as I’ve
 gotten into Forth, what I thought was a lack
of understanding of Forth on my part
turned out to be a lack of understanding of
how something goes on inside my com-
puter! I needed to see how interrupts work,
even on a simple level, before I could
resolve some file I/O problems I had, for
example. Forth handled my needs quite
well, once I knew what had to be done and
how to go about it. On a recent project
written in both Turbo BASIC and F83, I
reduced the program file size by 30%,
reduced execution time about 60%, and
made the source instructions much clearer
using the Forth system!

Algorithms! Get a nice, readable book
on data structures. Try implementing some
modules in a language more familiar to you
(BASIC, possibly), then again in Forth.
I’ll bet there’s a wealth of advice and sugges-
tions to be had in this vein here on GENie’s
Forth forum! Am I right everybody?

Category I, Topic 33, Message 30
From: NMorgenstern [Leonard]
To K.Smith: Yea and verily! Your
experience is that of many others. Mahlon
Kelly commented a few weeks ago on one of
our Figgy Bar sessions that computer
languages were designed to give the user
access to the computer but, more impor-
tantly, to protect the computer from the
user. This is accomplished, of course, by
limiting what the language can do. Forth is
free of these restrictions. Mahlon teaches
Forth, and students have told him that, for
the first time, they understand the com-
puter.

Helen Burke, a friend of mine who is a
well-known metal sculptor, talks a lot
about organic form in her art, meaning that the form grows naturally from the materials and the function. Forth is organic in this sense, growing from the microprocessor and operating system rather than from a preconceived set of rigid ideas about what a computer language should look like, à la Wirth, Kernighan, and others. Regrettably...

Category 2 Introduction to Forth, Topic 5
From: M.Silva (Forwarded)
Sub: for us beginners? Help

We beginners need a place to get our feet wet. I am somewhat of an accomplished programer in assembler, Fortran, COBOL, Pascal... but not Forth. Where do I get started?

Category 2, Topic 5, Message 106
From: C.Struyckcn!

Very basic question: I am trying to get condensed mode out of my stargemini10x using F83. In screen 44 of utility.blk, I changed this:

: epson
control 0 emit ;
to this:

: starcond
control 15 emit ;
and also replaced the noop in the next line with starcond. I then loaded the screen.

This does not seem to work. I tried to see if save-system f83.com would make a difference... it did not. How do I get this to work? I also noticed that the whole screen got reloaded, resulting in many "already exists" notices. Does this mean Forth has now two identical compilations of each of these "already exists" words? If so, what is the correct way to load a word without reloading the whole screen?

Category 2, Topic 5, Message 107
From: NMorgenstern [Leonard]
To: C.Struyckcn!

"I changed control 0 emit to. . . control 15 emit..."

In F83, the word CONTROL gets the next word from the input stream and masks its first character back to five bits. Thus, control-0 is the same as 15 (decimal), while control-15 would be 1.

On Epson printers and many others, 15 (control-o) should put you into condensed mode. But you have to send it to the printer.

You should type PRINTING ON first.

"I got a lot of 'already exists' messages."

Forth will warn if you are redefining a word. It is a warning, not necessarily an error, because sometimes you want to redefine something. In your case, it was an error. You need to FORGET the words you have defined. Thus, FORGET FOO removes the word FOO and all words subsequently defined. F83 makes forgetting easy by a special word, MARK. The first thing I do before loading anything is type MARK TODAY. Then, if I type TODAY, it forgets everything after TODAY, but not the word TODAY itself. F83 "makes it easy to forget," as the old song goes.

It sounds to me as if you are making good progress. Please keep asking questions—others learn from the answers, as well as yourself. Also, if you can, attend the Sunday night round tables. They are specially aimed at beginners like yourself. Good luck!

Category 2, Topic 5, Message 108
C.Struyckcn!
To: NMorgenstern

Thanks very much for your help and encouragement. I am just starting to work my way through chapter nine of Starting Forth (second edition), and things are becoming a lot more confusing. In the meantime, I still have not resolved the printer mystery. (I do have a condensed printout of all the blocks now by flicking the appropriate dip-switch on the printer.) This is what I have discovered so far: When the printer is not hardware-forced into condensed mode and I use the command PRINTING-ON, typing 15 EMIT or CONTROL 0 EMIT will software-force the printer into condensed mode. But, after having changed NOOP TO EPSON in the second line of screen 44, i.e.,

DEFER INITPR
  ' EPSON IS INIT-PR

the words PRINTING-ON, SHOW, and LISTING should set the printer in condensed mode by themselves (because they all use INIT-PR in their definition). I get the feeling they all are still referring to the old INIT-pr, before I changed it. Does F83 use a precompiled UTILITY.BLK, and does it just pretend it is loading the screens? When, as an experiment, I tried to

FIG-FORTH for the Compaq, IBM-PC, and compatibles. $35 Requires DOS 2.0 or later, uses standard DOS files, hard disk or floppy.

Full-screen editor uses 16 by 64 format, has HELP screen via single keystroke. Source included for editor and other utilities.

SAVE allows storing Forth with all currently defined words onto disk as a COM file.

Definitions are provided to allow beginners to use Starting Forth as an introductory text.

Source code available as option, add $20.

Metacompiler for 6303/6803 Runs on a host PC, produces a PROM for a target board. Includes source for 6303 FIG-FORTH with multi-tasker. Application code can be Metacompiled with Forth to produce a target application PROM. $280

Metacompiler for 68HC11 As above, except power failure handling is omitted $268

ALL CMOS Processor Board Utilizes the 6303. Size: 3.93 by 6.75 inches. Uses 11-25 volts at 12ma plus current for options. $175-225

Up to 24 kb memory: 8k RAM, 8k PROM, additional 8k RAM or PROM as desired. Backup of RAM via off board battery.
Serial port and up to 40 pins of parallel I/O. Processor buss available at optional header to allow expanded capability via your interface board.

Micro Computer
Applications Ltd
8 Newfield Lane
Newtown, CT 06470
203-426-6164

Foreign orders add $5 shipping and handling. Connecticut residents add sales tax.
FORGET the original EPSON, I got a "Below Fence" message. When does one need to do a SAVE-SYSTEM F83.COM, and why and when does one need to metacompile? Do these things all have to do with the fact that F83 is working under DOS?

Category 2, Topic 5, Message 109
From: Pete Koziar
Subj: printer initialization

One important step you left out: F83 does not invoke EPSON when printing; it invokes a deferred word called INIT-PR, which is set up to be a NOOP. To use that printer control, type:

`epson is init-pr`

before you try to print or list anything. If you then want a listing, just type list-ing any time after redirecting INIT-PR. If you just want to echo what is on the screen in condensed mode, you would need to say:

`printing {`

There is another word, by the way, known as PAGE. If your printer supports automatic form-feeds (most do, nowadays), you should also type:

`form-feed is page`

I hope this helps!
Via Qwikmail 2.01 The Baltimore Sun

Category 2, Topic 5, Message 110
From: C.Struycken
To: Peter Koziak
Subject: printer initialization

Thanks for your response, Peter. I had already reset NOOP to Epson, but this did not make it work either. I finally figured that the words in UTILITY.BLK must be precompiled and that, therefore, the other words that use init-pr in their definitions are using the older init-pr that was set to NOOP. Does this make any sense? Without really knowing what I was doing, I re-metacompiled the system and now everything is working. It is still not completely clear what the metacompiling does and how it differs from save-system, but maybe the "under the hood" chapter in Starting Forth will make things a bit clearer.

Category 2, Topic 5, Message 111
From: Steve Palincsar
Subject: F83 utility.blk

It's been several years since I seriously looked at F83, but as I recall you are absolutely correct in your surmise that it uses a precompiled UTILITY.BLK. All the .BLK files supplied in the .ARC file are there for documentation and have already been incorporated in the F83.COM file. I don't
recall any optional extension files in the Laxon & Perry package itself that you need to load. (There are, of course, hundreds if not thousands of extensions for F83 in the public domain.)

Category 2, Topic 5, Message 113
From: P. Sergeant
C. Struycken, there is no need to redefine the word EPSON or to put the new word in UTILITY.BLK next to EPSON. There is no need to recompile your Forth system. Leave EPSON alone and define a brand-new word that will initialize your printer, which I gather is not an Epson. Call the new word STARCOND, as you suggested, or GEMIN10X, or whatever. Put it anywhere in an empty screen.

You would want code something like this:

HEX
: GEMIN10X ( -- )
  15 EMIT ;

(This word puts the Gemini printer into condensed mode.)

' GEMIN10X IS INIT-PR
(This re-vectors INIT-PR so it will use your brand-new definition when it does SHOW, etc., without having to recompile your Forth system.)

SAVE-SYSTEM F83G.COM
(The "G" is to remind you that this version will work with your Gemini printer.)

I hope this clears things up.—Frank

* * * *

In the next "Best of GEnie" column, we will look at how ForthNet has grown since its first faltering steps a few months ago.

Many of the messages above were posted to GEnie via ForthNet, thanks in large measure to the unflagging efforts of Jerry Shifrin, sysop of the East Coast Forth Board.

To suggest an interesting guest, please leave e-mail posted to GARY-S on GEnie (gars on Wetware and the Well), or mail me a note. I encourage anyone with a message to share to contact me via the above or through the offices of the Forth Interest Group.
Forth Interest Group

The Forth Interest Group serves both expert and novice members with its network of chapters, Forth Dimensions, and conferences that regularly attract participants from around the world. For membership information, or to reserve advertising space, contact the administrative offices:
Forth Interest Group
P.O. Box 8231
San Jose, California 95155
408-277-0668

Board of Directors
Robert Reiling, President (ret. director)
Dennis Ruffer, Vice-President
John D. Hall, Treasurer
Terri Sutton, Secretary
Wil Baden
Jack Brown
Mike Elola
Robert L. Smith

Founding Directors
William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge
John James

In Recognition
Recognition is offered annually to a person who has made an outstanding contribution in support of Forth and the Forth Interest Group. The individual is nominated and selected by previous recipients of the "FIGGY." Each receives an engraved award, and is named on a plaque in the administrative offices.

1979 William Ragsdale
1980 Kim Harris
1981 Dave Kilbridge
1982 Roy Martens
1983 John D. Hall
1984 Robert Reiling
1985 Thea Martin
1986 C.H. Ting
1987 Marin Ouverson
1988 Dennis Ruffer

ANS Forth
The following members of the ANS X3J14 Forth Standard Committee are available to personally carry your proposals and concerns to the committee. Please feel free to call or write to them directly:

Gary Betts
Unisyn
301 Main, penthouse #2
Longmont, CO 80501
303-924-9193

Mike Nemeth
CSC
10025 Locust St.
Glenndale, MD 20769
301-286-8313

Andrew Kobziar
NCR Medical Systems Group
950 Danby Rd.
Ithaca, NY 14850
607-273-5310

Elizabeth D. Rather
FORTH, Inc.
111 N. Sepulveda Blvd., suite 300
Manhattan Beach, CA 90266
213-372-8493

Charles Keane
Performance Packages, Inc.
515 Fourth Avenue
Waterleit, NY 12189-3703
518-274-4774

George Shaw
Shaw Laboratories
P.O. Box 3471
Hayward, CA 94540-3471
415-276-5953

David C. Petty
Digitel
125 Cambridge Park Dr.
Cambridge, MA 02140-2311
617-576-4600

Forth Instruction
Los Angeles—Introductory and intermediate three-day intensive courses in Forth programming are offered monthly by Laboratory Microsystems. These hands-on courses are designed for engineers and programmers who need to become proficient in Forth in the least amount of time. Telephone 213-306-7412.

On-Line Resources
To communicate with these systems, set your modem and communication software to 300/1200/2400 baud with eight bits, no parity, and one stop bit, unless noted otherwise. GENie requires local echo.

GENie
For information, call 800-638-9636
- Forth RoundTable (ForthNet link*)
  Call GENie local node, then type M710 or FORTH
  SysOps: Dennis Ruffer (D.RUFFER), Scott Squires (S.W.SQUIRES), Leonard Morgenstern (NMORGENSTERN), Gary Smith (GARY-S)
- MACH2 RoundTable
  Type M450 or MACH2
  Palo Alto Shipping Company
  SysOp: Waymen Askey (D.MILEY)

(Continued on next page)
We want the Forth community to know that their efforts are greatly appreciated here.

Yours faithfully,
Lance Collins, Secretary
Melbourne Chapter

(Reference Section continued)

BIX (ByteNet)
For information, call 800-227-2983
• Forth Conference
Access BIX via TymeNet, then type
j forth
Type FORTH at the : prompt
SysOp: Phil Wasson (PWASSON)
• LMI Conference
Type LMI at the : prompt
Laboratory Microsystems products
Host: Ray Duncan (RDUNCAN)

CompuServe
For information, call 800-848-8990
• Creative Solutions Conference
Type IGo FORTH
• Computer Language Magazine Conference
Type IGo CLM
SysOps: Jim Kyle, Jeff Brenton, Chip Rabinowitz, Regina Starr Ridley

UNIX BBS’s with Forth conferences (ForthNet links*)
• WELL Forth conference
Access WELL via CompuServeNet or 415-332-6106
Fairwimess: Jack Woehr (jax)
• Wetware Forth conference
415-753-5265
Fairwimess: Gary Smith (gars)

PC Board BBS’s devoted to Forth (ForthNet links*)
• East Coast Forth Board
703-442-8695
SysOp: Jerry Schifrin
• British Columbia Forth Board
604-434-5886
SysOp: Jack Brown
• Real-Time Control Forth Board
303-278-0364
SysOp: Jack Woehr
• Melbourne FIG Chapter
Lance Collins
(03) 299-1787 in Australia
61-3-299-1787 international

(Letters, continued from page 6)

CASE will increment this count:

: +CASE ( -- )
4 S> 4 S>
1+
4 >S 4 >S ;

: CASE ( n -- )
+CASE
4 S@ = IF ;

Finally, END will do all of the cleanup:

: FORCE ( -- )
3 S> DROP
1 3 >S ;

: END ( -- )
FORCE
4 S> DROP
4 S>
0 DO THEN LOOP ;

CASE is no longer needed, as its function has been absorbed by END.

With these changes, the case statement can help protect a programmer from an oversight or a miscount. The disadvantages here are some additional overhead in CASE and a larger case stack.

Enjoy,
Wes Cowley
P.O. Box 280138
Tampa, Florida 33682-0138
wcowley@dci2wc.das.net or wes@cup.portal.com

On-line Down Under
Dear Editor,

The Melbourne Chapter of the Forth Interest Group wishes to acknowledge the support we have had in keeping our chapter going and in setting up our bulletin board.

We wish to thank Robert Reiling for his encouragement and help in obtaining an early copy of F-PC for us, and some other Forth software to start our board with last year.

We particularly thank Jerry Schifrin for his initial donation of files, which really gave our members something to think about. Recently, we have had another large batch of files from Jerry, which makes our board a major resource for Forth people here.

—Marlin Ouverson
Editor

We want the Forth community to know that their efforts are greatly appreciated here.

Yours faithfully,
Lance Collins, Secretary
Melbourne Chapter

(REFERENCE SECTION continued)
The FIG Chapters listed below are currently registered as active with regular meetings. If your chapter listing is missing or incorrect, please contact Kent Safford at the FIG office's Chapter Desk. This listing will be updated in each issue of Forth Dimensions. If you would like to begin a FIG Chapter in your area, write for a "Chapter Kit and Application." Forth Interest Group, P.O. Box 8231, San Jose, California 95155

**U.S.A.**

- **ALABAMA**
  Huntsville Chapter
  Tom Konantz
  (205) 881-6483

- **ALASKA**
  Kodiak Area Chapter
  Ric Shepard
  Box 1344
  Kodiak, Alaska 99615

- **ARIZONA**
  Phoenix Chapter
  4th Thurs., 7:30 p.m.
  Arizona State Univ.
  Memorial Union, 2nd floor
  Dennis L. Wilson
  (602) 381-1146

- **ARKANSAS**
  Central Arkansas Chapter
  Little Rock
  2nd Sat., 2 p.m. &
  4th Wed., 7 p.m.
  Junjkind Photo, 12th & Main
  Gary Smith (501) 227-7817

- **CALIFORNIA**
  Los Angeles Chapter
  4th Sat., 10 a.m.
  Hawthorne Public Library
  12700 S. Grievea Ave.
  Phillip Wasson
  (213) 649-1428

  North Bay Chapter
  2nd Sat., 10 a.m.
  Forth, AI
  12 Noon Tutorial, 1 p.m.
  FortSouth Berkeley Public Library
  George Shaw (415) 276-5953

  Orange County Chapter
  4th Wed., 7 p.m.
  Fullerton Savings
  Huntington Beach
  Noshir Jesung (714) 842-3032

  Sacramento Chapter
  4th Wed., 7 p.m.
  1708-59th St., Room A
  Tom Ghomley
  (916) 444-7775

  San Diego Chapter
  Thursdays, 12 Noon
  Guy Kelly (619) 454-1307

  Silicon Valley Chapter
  4th Sat., 10 a.m.
  H-P Cupertino
  Bob Barr (408) 435-1616

  Stockton Chapter
  Doug Dillon (209) 931-2448

- **COLORADO**
  Denver Chapter
  1st Mon., 7 p.m.
  Clifford King (303) 693-3413

- **CONNECTICUT**
  Central Connecticut Chapter
  Charles Krajewski
  (203) 344-9996

- **FLORIDA**
  Orlando Chapter
  Every other Wed., 8 p.m.
  Herman B. Gibson
  (305) 855-4790

  Southeast Florida Chapter
  Coconut Grove Area
  John Forsberg (305) 252-0108

  Tampa Bay Chapter
  1st Wed., 7:30 p.m.
  Terry McNay (813) 725-1245

- **GEORGIA**
  Atlanta Chapter
  3rd Tues., 6:30 p.m.
  Western Sizzlen, Doraville
  Nick Hennenfent
  (404) 393-3010

- **ILLINOIS**
  Chicago Chapter
  4th Sat., 10 a.m.
  H-P Cupertino
  Bob Ban (408) 435-1616

  Central Illinois Chapter
  Champaign
  Robert Illyes (217) 359-6039

- **INDIANA**
  Fort Wayne Chapter
  2nd Tues., 7 p.m.
  IP Univ. Campus, B71 Neff Hall
  Blair MacDermid (219) 749-2042

- **IOWA**
  Central Iowa FIG Chapter
  1st Tues., 7:30 p.m.
  Iowa State Univ., 214 Comp. Sci.
  Rodrick Eldridge
  (515) 294-5659

- **MARYLAND**
  MDFIG
  Michael Nemeth
  (301) 262-8140

- **MASSACHUSETTS**
  Boston Chapter
  3rd Wed., 7 p.m.
  Honeywell
  300 Concord, Billerica
  Gary Chanson (617) 527-7206

- **MICHIGAN**
  Detroit/Ann Arbor Area
  4th Thurs.
  Tom Chrapkiewicz
  (313) 322-7862

- **MINNESOTA**
  Minnesota Chapter
  Minneapolis
  Fred Olson
  (612) 588-9532

- **MISSOURI**
  Kansas City Chapter
  4th Tues., 7 p.m.
  Midwest Research Institute
  MAG Conference Center
  Linus Orth (913) 236-9189

  St. Louis Chapter
  1st Tues., 7 p.m.
  Thornton Branch Library
  Robert Washam
  91 Weis Drive
  Ellisville, MO 63011

- **NEW JERSEY**
  New Jersey Chapter
  Rutgers Univ., Piscatawy
  Nicholas Lordi
  (201) 338-9363
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Location</th>
<th>Contact Person</th>
<th>Address/Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW MEXICO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Albuquerque Chapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEW YORK FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rochester Chapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Columbus FIG Chapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peninsula FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detroit FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Philadelphia FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Jersey FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Northern Ohio FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Southern Ohio FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pensacola FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pennsylvania FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pittsburgh FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIRGINIA FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VIRGINIA FIRST FORTH OF HAMPTON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROY, W. W. Restaurant</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OREGON</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Willamette Valley Chapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PENNSYLVANIA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Villanova University</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TENNESSEE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>East Tennessee Chapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXAS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUSTIN FIG</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** INTERNATIONAL **

<table>
<thead>
<tr>
<th>Country</th>
<th>Chapter</th>
<th>Location</th>
<th>Contact Person</th>
<th>Address/Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUSTRALIA</td>
<td>Melbourne Chapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENGLAND</td>
<td>Forth Interest Group-UK</td>
<td>London</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FINLAND</td>
<td>FinFIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOLLAND</td>
<td>Holland Chapter</td>
<td>Vic Van de Zande</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITALY</td>
<td>FIG Italia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAPAN</td>
<td>Japan Chapter</td>
<td>Toshi Inoue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORWAY</td>
<td>Bergen Chapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPUBLIC OF CHINA</td>
<td>R.O.C. Chapter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWEDEN</td>
<td>SweFIG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWITZERLAND</td>
<td>Swiss Chapter</td>
<td>Max Hugelshofer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECIAL GROUPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC4000 Users Group</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Volume XI, Number 3* 39 *Forth Dimensions*
FORML CONFERENCE

The original technical conference
for professional Forth programmers, managers, vendors, and users.

Following Thanksgiving, November 24–26, 1989

Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California U.S.A.

Forth and Object-Oriented Programming

Papers are invited that address relevant issues in the development of object-oriented programming and object-oriented applications. Data structures to support object-oriented program development are readily constructed in Forth. These structures may be reused which increases productivity when new applications are developed. Papers about other Forth topics are also welcome.

Mail your abstract(s) of 100 words or less to FORML Conference, Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.

Completed papers are due November 1, 1989.

Conference Registration

Registration fee for conference attendees includes conference registration, coffee breaks, and note-book of papers submitted, and for everyone rooms Friday and Saturday, all meals including lunch Friday through lunch Sunday, wine and cheese parties Friday and Saturday nights, and use of Asilomar facilities.

Conference attendee in double room—$285 • Non-conference guest in same room—$150 • Children under 17 in same room—$100 • Infants under 2 years old in same room—free • Conference attendee in single room—$335

Register by calling the Forth Interest Group business office at (408) 277-0668 or writing to: FORML Conference, Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.

Forth Interest Group
P.O.Box 8231
San Jose, CA 95155