$10 Volume XV, Number 1 May 1993 June

FORTH DIMENSIONS

Forth in Search of a Job

Integer Date Calculations

Calendars & The Game of Life

Interface for the GPIB (IEEE-488)

Formatted Input Fields

Contents

Features

Departments

Editorial

With this issue, we begin our fifteenth year of publication, an anniversary of sorts. Looking back, the times have certainly changed, but has Forth? We certainly are no longer as separatist—it took years for many in our community to reluctantly accept a vision of Forth that would allow it to run under another operating system, even (gasp) using standard disk formats. But despite those and perhaps more fundamental changes, Forth retains its unique signature, its forté undiminished. And although C firmly squashed any dreams of becoming the next Pascal, Forth does occupy a small but apparently stable niche.

Will our language of choice ever become others' language of necessity? Should it evolve in more revolutionary than incremental ways? I suspect that ANS Forth will prove to be only the next shaping influence, and possibly not the most powerful, of many we will encounter.

Those still harboring dreams of grandeur would do well to consider the (inherent?) obstacles to Forth's breakthrough into broader arenas. One author in this issue has done that. Donald Kenney, who describes himself as a "reformed software manager," bluntly enumerates in his "Forth in Search of Work" the items many managers see as stumbling blocks.

Contest for Authors

Not unrelated to the above is this year's FORML Conference theme and the subject of *FD*'s current contest. "Forth Development Environments" encompasses a wide range of possible topics, and we look forward to hearing your take on the subject. See the contest announcement on page six (which lists the *cash prizes* the winners will receive and the *contest deadline*) and the FORML ad on the back cover for just a few ideas. We welcome any and all submissions that address the general contest theme, from the mundane to the exotic. Take this opportunity to get involved *and* get paid!

Public Defender

Computers in Physics (500 Sunnyside Blvd., Woodbury, NY 11797) published an article about programming languages and tools which neglected to mention Forth. Fortunately, Edgar T. Lynk—their reader and ours—took steps to correct that oversight. His prompt letter appeared in their January/February 1993 issue, reminding them of Forth's ubiquity across many CPUs, its object-oriented characteristics, and its availability in both professional and public-domain implementations. And his letter included the current address and telephone number of the Forth Interest Group.

As part of our Author Recognition Program, we will be sending Edgar a certificate redeemable for his next year's membership in FIG. Efforts like his, which inform the broader community of Forth's continued vigor and relevance, are crucial to keeping Forth in the public's awareness and to telling the uninformed about the Forth Interest Group. In the absence of corporate mega-funding and widespread adoption by universities (remember C's beginnings?), we appreciate and encourage grass roots actions that, simply and directly, keep Forth part of the mainstream dialog.

Wit's End

In a letter accompanying his latest submission, Charles Curley suggests that, if I am going to print bad puns, I should include Charlie Johnsen's observation that Forth is a very libertarian language—where else can you say, "STATE OFF"?

—Marlin Ouverson

Forth Dimensions
Volume XV, Number 1
May 1993 June

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Frank Hall

Forth Dimensions welcomes editorial material, letters to the editor, and comments from its readers. No responsibility is assumed for accuracy of submissions.

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $40 per year ($52 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 2154, Oakland, California 94621. Administrative offices: 510-89-FORTH. Fax: 510-535-1295. Advertising sales: 805-946-2272.

The Forth Interest Group

The Forth Interest Group is the association of programmers, managers, and engineers who create practical, Forth-based solutions to real-world needs. Many research hardware and software designs that will advance the general state of the art. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $40/46/52 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 2154, Oakland, CA 94621."

Letters

Letters to the Editor—and to your fellow readers—are always welcome. Respond to articles, describe your latest projects, ask for input, advise the Forth community, or simply share a recent insight. Code is also welcome, but is optional. Letters may be edited for clarity and length. We want to hear from you!

Of Saints and Spaghetti

Dear Mr. Ouverson,

Forth Dimensions XIV/6 is great. Tim Hendtlass asks, "Math—Who Needs It?" I do, I do! The Laxen & Perry F83 I use runs a bit short of math power when it comes to double and float routines. I don't need them all the time, but it's good to have them in the toolbox when I do. Looking forward to [the rest of the code].

Gordon Charlton's "Unified Control Structure" seems to be a case where too much simplicity actually creates complexity. By having the control structures use differently named words—e.g., `IF THEN` or `IF ELSE THEN` for branches, and `BEGIN AGAIN`, `BEGIN WHILE REPEAT`, `BEGIN UNTIL`, or `DO LOOP`, `DO +LOOP` for loops—the type and function of the control structure is more obvious than if you try to use fewer words in endless permutations.

I could see a need for something like `WHEN` in the `BEGIN AGAIN` loop, which would allow for multiple exits from the loop. Right now I would use `EXIT`, but the contruct isn't perfect. Might also have a `?WHEN` and a `?0WHEN` to exit the loop on true or false flags on the stack. In that regard, multiple `WHILE` and `UNTIL` statements would be more confusing to me. These loops are better left in their classic form.

Charles Moore's "Fireside Chat": Forth as Zen! Just when I was beginning to understand Forth, Moore comes up with spaghetti code written in machine language as the new way in Forth. Saints preserve us, Paddy! I hope you can implore C.H. Ting to expand the explanation of what goes on in 386 OK.

Yours truly,
Walter J. Rottenkolber
P.O. Box 1705
Mariposa, California 95338

Smell the Earth, Hear the Seagulls

Dear Mr. Ouverson,

Mike Leavitt, Utah's new governor, tells an anecdote about his grandfather, a farmer. His grandfather made do with an old tractor for years and years, while his neighbor on the farm next door always had to have the latest time-saving farm implement. When asked about his philosophy, the grandfather said, "I like to stick with what's real and right."

Over time, the neighbor went broke paying for his equipment, while Mike's grandfather thrived. This story has struck such a chord with Utahns that "real and right" has become a motto and a budget criterion for the state.

To me, Forth is "real and right." It may not be glitzy, and sometimes I curse its quirks just as Mike's grandfather must have done with his old tractor; but when it comes to doing real work at reasonable cost, it can't be beat. There's a definite advantage to having simple tools that you know extremely well.

It's difficult to justify this attitude in the current software community's attitude that more powerful tools are the answer to our programming dilemmas. Strong voices, such as that of Tyler Sperry, editor of *Embedded Control* magazine, have called for Forth to catch up with the world and implement all the popular programming tools. To an extent, I agree with these people. I prefer to program in F-PC Forth, which is considered by many to be a "fat Forth," with more tools than I really need. There are times when these features come in really handy, but as I analyze my programming practices, the more I use Forth, the more I find that I use a very small, core set of tools and techniques to implement extremely powerful functionality.

I contrast this with the attitude at my work, where the latest CASE tools, emulators, processors, and C compilers must be used in the products we design. The cost of doing this has been much more expensive development, higher product cost, lower processor efficiency (which translates to sluggish product performance), longer development times, and (still hard to believe, but true) reduced portability. The single benefit I have seen from it so far is that the generated software has had higher quality in terms of bugs found after product release, as compared with previous assembly language development.

I think the call for more powerful tools and standards is justified in order to make Forth more friendly to new users. I'm even looking forward to the 32-bit, flat address space Windows NT Forths that must certainly be in development. But I hope the implementors make efforts to provide paths and incentives for new users to move naturally to the "real and right" Forth that places the emphasis on results instead of on tools. The power of tools is often mistakenly perceived as form (windows, colors, graphics, etc.) rather than substance (what you can do with it). `CREATE DOES>` is the most powerful tool I've seen in any language, including object-oriented ones.

I think if I were a farmer, I'd rather smell the earth and hear the seagulls than sit way up in an air-conditioned cab listening to heavy metal.

Thanks for your efforts,
Glen Dixon
587 N. 2575 W.
West Point, Utah 84015

Roving Reporter at Silicon Valley

For the longest time, I just didn't give a FIG! That's right, it had been 15 years since my last FIG experience. Ah, gone are the days of Bill Ragsdale's original FIG meetings in Hayward, California—no more bulging programmer bellies stuffed inside silly FIG T-shirts. Well, at least most of the T-

shirts are gone...

Saturday's Silicon Valley FIG Chapter meeting was a fascinating experience that could, of course, only occur in a room full of 40–50 Forth programmers. The format was pretty much the same as the early days: the first part of the day-long affair was directed toward "nuts-and-bolts" Forth issues, the next part for "stand up and introduce yourself or plead your case," then a guest speaker, an informal discussion break, and then the Zen period. That's right, Zen. More on that later.

Andrew delivered a scintillating overview of using text files instead of those pesky disk screens/blocks for dealing with Forth text. Blocks are simple and easy, and are the simplest and easiest method to implement on new hardware. But, take it from this author, if you have to support any large Forth system that someone else has written, blocks are a nightmare! You'll always have to re-document the whole damn thing to figure out what's going on.

Before Andrew knew what was going on, a heated debate broke out among audience members over the specific definition of a "line"! Just what does a line end with: <CR>? <FF>? the 80th character? After the shooting was over, there were only three casualties. Andrew did a great job, and even provided printouts of his code, a disk containing the code (!), and free balloons for the kids.

Next, Shannon cruised by for an overview of object-oriented programming in Forth. His company did not just dump everything and convert to Oops, they only went Oops when needed. As explained by Shannon, this seemed a most practical approach. A general discussion around the room concluded that the best way for Forth programmers to get familiar with Oops is to convert something they've already written—something which lends itself to Oops. There were many harrumphs in support of this. Oops, like riding a bicycle, seems difficult to teach. A programmer simply needs to sit down and do Oops. I suppose they could even do Oops loops... *not!*

The final word regarding object-oriented programming was a blistering comment by someone with a pony tail, who shall remain nameless because I don't know his name. He observed, to the general agreement of our august, duly assembled Forth body, that more money has probably been made *selling* Oops languages (C++, for instance) than has been made by *programming* in Oops. There was the typical roar of iconoclastic Forth programmer laughter. (Plus a few belches.)

Speaking of belches, about this time (high noon) the Great Pizza Wizard arrived bearing several of those tell-tale, flat, white boxes with thick grease stains on the bottom.

Next was the stand-up-and-be-counted session in which

(Continues on page 9.)

Formatted Input Fields

Martin Schaaf
San Mateo, California

Writing the user interface is almost always the most tedious part of any project. Tools that will prompt for input, format input and output, and automatically check for correct input, make the job much easier. Allowing the user to edit input strings by backspacing over them makes for a "user friendly" interface.

The following >PICTURE tool was inspired by the PICTURE statement in Cobol. (I dropped out of the Cobol course when one error produced approximately 50 error messages!) In my version, I split the function into input and output words. I can then use data in a normal Forth manner, and worry about formatting only during input and output. At the end of this article is an example of formatted input of a phone number, followed by conversion to a double-precision number on the stack.

```
create task variable p-test     \ Used to save the picture type operator.
variable echo                        \ Flag if input is to be echoed.
: bs's 0 do 8 emit loop ;            \ Backspace over a field on the crt.

: alpha dup 64 > over  98 < and over         \ Check for characters that
  dup 96 > swap 123 < and or                 \ might be used in an
  over ascii - = or  over ascii " = or       \ ALPHA field
  over ascii : = or  over ascii + = or       \ True if found
  over ascii & = or  over ascii ; = or       \ False if not found
  over ascii $ = or  over ascii ' = or
  over ascii . = or  over bl = or ;

: alpha-numeric dup 31 > over 127 < and ;    \ Ditto for ALPHA-NUMERIC
: numeric       dup 47 > over  58 < and ;    \ Ditto for NUMERIC fields


: bl-mask                       0 do         \ Mask out the underline
  2dup swap i + c@ dup                       \ characters while moving
  ascii _ = if                               \ the picture to the data
  drop bl then                               \ field. This way, if the
  swap i + c! loop 2drop ;                   \ user presses [ENTER]
                                             \ before the end of the
                                             \ field, the remainder of
                                             \ the field will be blank.

: ?echo echo @ if dup else ascii * then emit ;     \ I use this to turn off
                                                   \ echoing on passwords

: ?bs's  ( D P I1 - D P I2 )        0 over 1 - do
  1 < if 7 emit 0 leave else
  2dup i tuck + c@ dup 8 emit emit 8 emit
```

(Code continues on next page.)

(Code continued from previous page.)

```
  dup ascii _ = if drop bl -rot + c! i leave else
  -rot + c! i then then -1 +loop ;

\ When the user presses the backspace key, you have to backspace
\ over prior input but leave formatting characters intact. I also
\ leave the revised index on the stack, to be used to decrement
\ the loop counter in (>picture).
```

The following run-time word performs the work of pictured input, using the above subwords.

```
: (>picture) ( i - )       pad +       \ Address of the data area
  r> dup @ p-test !                    \ Save the field type operator
  2+ count 2dup + >r                   \ Get the picture field string
  2dup tuck type                       \ Display it with underlines
  bs's                                 \ Position cursor at beginning
  3dup rot swap bl-mask 0 do           \ Write the picture to data area
  2dup i + c@ dup ascii _ - if         \ If character of picture is not
  tuck emit i + c! 1 else              \ an underline, emit it
  drop key dup 13 = if                 \ If input character is [RETURN]
  2drop leave else dup 8 = if          \ exit, otherwise if BS
  2drop i ?bs's i - else               \ BS to beginning of field
  p-test perform if                    \ otherwise test membership
  ?echo swap i + c! 1 else             \ conditionally echo then store
  2drop 7 emit 0 then                  \ Beep on illegal character or
  then then then +loop                 \ end of line
  echo on 2drop ;                      \ Turn on echo for next use...
```

The >PICTURE operator is a compiler word. All work that can be done ahead of time is done while compiling. The usage is:

```
NN >PICTURE TYPE " ___-____"
```

where NN is the offset into pad of the field and TYPE is the ALPHA, NUMERIC, or ALPHA-NUMERIC type word (other special type words could be defined). " is the usual delimiter character, though other delimiters could be used.

___-___-____ is the picture string—non-underline characters will be preserved in the final input (in this case, a telephone number).

```
: >picture      compile (>picture)    \ Compile the picture operator
  [compile] [compile]                 \ Compile the following type
  bl word 1+ c@                       \ Get the delimiter character
  word c@ 1+ allot ;                  \ and enclose the picture string
  immediate                           \ It's a compiler word, so it
                                      \ has to run immediately
```

A typical usage of this tool would be:

```
5 >PICTURE NUMERIC " ___-___-____"    \ Get the data in the field
5 pad + phone-swap                    \ Swaps the first two digits of the area code to keep
                                      \ double number precision
0.0 4 pad                             \ convert the phone number into
convert convert convert drop          \ double number on the stack
```

The same type of tool could be used for pictured output (in this case, a social security number):

```
nn PICTURE> TYPE " ___-__-____"
```

I leave `PICTURE>` as an exercise for the reader.

Having completed this tool, I can code user input and output much faster. There's just one problem: It's now much more boring!

Martin Schaaf is a Forth programmer in San Mateo, California. In the early 1980s, he initiated the FORTH-BIT discussion group, which inspired a number of hardware implementations of Forth. Mr. Schaaf currently is the sysop of "the BBS from 38 NORTH - 122 WEST" and can be reached at:

Voice: 415-344-8820
BBS: 415-343-7517, v.32bis
NETMAIL: 1:125/121

(Letters, continued from page 6.)

everyone introduced themselves. However, no one stood up. Too much pizza all around, I suppose. At this point, this author was harshly interrogated regarding his successful contract at the capital airport of Saudi Arabia, where he managed the largest Forth control system in the world for over two years, and also learned how to ferment his own wine. This enormous control system is a network of 700 polyFORTH computers interconnected via high-speed hardware and the ClusterForth layer to allow easy programmer and software tie-in to any other computer on the network.

The control system can monitor and control 26,000 points, 800 card readers, and 12,000 employees. It is database driven, and the current database is 53 megabytes. One guy works full time just maintaining that humongous database—except five times a day, when he goes to pray. He probably prays for database integrity a lot... Allaht? *Not!*

Working in Saudi Arabia is a real adventure. However, I've seen too many marriages break up there to recommend it to anyone except those with a little bit of the crusader in them. Basically, you must ask yourself, "Self, how long will my wife put up without basic American freedom? She'll typically give you *her* answer the first night she's there!

Speaking of the fair sex, I was shocked to see that there was not one woman at the FIG meeting! How odd. There are women working all over the San Francisco Bay area, in all walks of life, in numbers almost equal to men. I've worked with several lady programmers through my years. And everyone's heard of Liz Rather! But at the Silicon Valley FIG Chapter? No females. Maybe the guys there have a reputation, I don't know. Maybe there's something about Forth that attracts men but repels women—sort of like The Three Stooges. Nyuk, nyuk!

Next, a fairly lucid young fellow went over the current DNA analysis and cloning techniques, 'cause that's what the company does that hosted our meeting. I particularly enjoyed (even waking up for) his ruthless political lambasting of Proxmire's Golden Fleece award regarding wasting money on DNA genoming. Anyway, each time the lecturer mentioned how much DNA acted like software, many of the Forth programmers squirmed excitedly in their seats.

Next, Dr. Ting, our illustrious leader, gave a revelatory treatise on "Zen is to other religions as Forth is to other software." Or was it the other way around? Well, I guess it doesn't matter from a yin/yang perspective. Or does it? Anyway, it was a fascinating story about many of the characters responsible for Chinese Zen. There were even some keen observations and commentary from the audience.

For instance, Ting revealed that the Buddha is within all of us. It is a special feeling or situation within, perhaps akin to one cup of coffee too many. Anyway, someone asked, if it's within us all, is it within Forth? Many suspected the interpreter...

Then there was that old "Is man essentially good?" thing, to which someone quipped, "Naah, he's a selfish mofo!" To which the good Dr. Ting corrected, "Selfish genes!" Fortunately, there were no Genes in the audience.

It was generally agreed that Charles Moore was fer sure an avatar and would make a bodacious Zen master. Unfortunately, though expected, Chuck did not show up and missed his chance for world-wide fame, fortune, and, of course, a gift mantra (we all pitched in).

Next, Dr. Ting presented his work on a book, each page of which contains English, Chinese, and Forth, all three saying the same thing—sort of an engineer's Rosetta Stone. A motion was made to bury a copy of this book, for future generations to dig up and revere. Someone cleverly suggested that Leo Brodie illustrate the new Gospel/Sutra. Dr. Ting's book was to be titled *EFORTH.* Someone explained to me this stands for "Evangelical Forth."

What a fun meeting it had been! How informative! And I was warmly pleased to see that the humor and spirit of the Forth programmer lives on despite Oops, recessions, and, especially, 386OK!

But then it grew late, we were all hungry, and I heard a Miller Draft calling my name from far across the San Mateo bridge...

—Reporting from the Silicon Valley, this has been your roving Forth programmer, Doug Bell. Internet: dougbell@netcom.com

Interface for the GPIB (aka IEEE-488)

Robert Thompson
Hattiesburg, Mississippi

The general purpose interface bus (GPIB) is a link or interface system through which interconnected electronic devices communicate. The original GPIB was designed by Hewlett-Packard (where it is called the HP-IB) to connect and control programmable instruments manufactured by Hewlett-Packard. Because of its high data transfer rate of from 250 kilobytes to one megabyte per second, the GPIB quickly gained popularity in other applications, such as intercomputer communication and peripheral control. It was later accepted as the industry standard IEEE-488. The versatility of the system prompted the name General Purpose Interface Bus.

National Instruments expanded the use of the GPIB among users of computers manufactured by companies other than Hewlett-Packard. They design hardware interfaces and software that helps users bridge the gap between their knowledge of instruments and computer peripherals, and of the GPIB itself.

National Instruments provides software interfaces to its GPIB handler in several languages (BASIC, Pascal, C, etc.) except Forth. Since Forth is frequently used for instrument control, it would be useful to have an interface written in Forth.

In our laboratory, we are using a frequency counter to acquire data in real time from a piezo-electric quartz crystal used as a mass-to-frequency transducer. Some biochemical and electrochemical reactions are accompanied by mass changes large enough to be detected when they are carried out on the surface of the crystal. We are using an IBM AT clone computer, a GPIB-PCII board from National Instruments, a model PM6680 frequency counter from Philips, and crystal oscillating circuitry built in our laboratory. The Forth system being used is F-PC v3.53, written by Tom Zimmer. This article will describe how an interface to Forth was written and how it has been used to acquire frequency data.

There are several interesting points I would like to make. The first is how to use the DOS I/O control interrupt to access the device driver. The second point, and I think the most important, is how to call this device driver from within the Forth environment. This is the part of the program that took the most time and effort to work out. Since National Instruments does not document the parameters needed for a successful call, I had to figure this out by using a little creative reverse engineering and an assembly language debugger. The third point is that one must be careful to preserve registers that Forth uses around this call to the device driver. These are documented in the F-PC manual, but there is nothing like a concrete example to drive the point home. The fourth point illustrates how to use Forth to transfer information across the GPIB. Finally, I have implemented a neat feature using the multitasking part of F-PC: real-time display of data and error messages from the frequency counter. I have written words which can send messages to a device on the GPIB when Forth is interpreting and compiling. When an error occurs, the counter displays an error number on its display. At the same time, it can be set to assert the service request line of the GPIB. Forth can be programmed to detect this, and one of my routines will get the error message from the counter and display it on the screen. This is a background processing loop that runs as long as F-PC is running. Another background loop will collect data from the counter and save it to a text file. A third background loop will display this data on the screen in real time. I think these are good examles of the multitasking abilities of Forth.

System Description

The GPIB handler is implemented by National Instruments as a CONFIG.SYS handler. This Forth program is an interface to the National Instruments GPIB handler. 30 commonly used GPIB funcitons can be accessed to control devices on the bus. The function codes are given as constants in the source code. Forth words are provided to print and decode the status variables set by the GPIB handler, to decode errors and print messages, and to call the GPIB handler. Forth words are also presented which acquire data from the Philips timer/counter and save it to a text file which can later be imported into a spreadsheet program.

The handler is installed in the computer at boot-up time by MS-DOS as a CONFIG.SYS driver, and its location in memory is returned to the Forth program by using the DOS IOCTL function. The heart of the application, then, is the Forth word which calls this handler, passing the GPIB function number to it, and returning with data and status information.

After the Nationl Instrument GPIB handler has been installed, the first thing Forth must do is get the address of the handler for subsequent calls to it. This is done by the Forth word `IOCTLREAD`, which is coded in assembly. This word uses the DOS IOCTL function $44, sub function $2 (read from device). The parameters passed to `IOCTLREAD` are:

nbytes ..... The number of bytes to read. In our case, this number is not used, so we pass in zero.

adr-buf ..... This is the address of the buffer in which the segment and offset of the GPIB handler will be returned. The device descriptor will be returned in this buffer also. The device descriptor is a temporary identification number used by the handler, and is not related to the GPIB address.

device_name .. This is the buffer containing the symbolic device name. The names of devices are dev0, dev1, ..., dev15. Sixteen devices can be connected to each GPIB board. In our case the name of the Philips frequency counter is dev10. The frequency counter has a set of dip switches in back to set the address of the device. Each device connected to a GPIB board must have a unique address.

To initialize a device, call IBFIND to open each device you will use. The stack action for IBFIND is (buf-adr -- descriptor), where buf-adr is the buffer containing the symbolic device name, and descriptor is an integer value returned by the handler. Save this descriptor and use it in subsequent function calls. If IBFIND fails to find the device, it will abort and print an error message saying that the file has not been found. If this occurs, the program should check to see that the correct device name has been passed to it.

The Forth word CALLGPIB is the keystone of this application. It actually passes program control to the handler. After the handler has performed its function, it returns control to Forth. All registers used by the device driver, as well as by F-PC, must be saved around this call; in this case, these are bp, ds, es, and si. The first two parameters passed to CALLGPIB (adr-buf, cnt) will vary, depending on the function call; the other parameters are needed in every case. It is only necessary for the programmer to supply the parameters, descriptor, and GPIB function. The other parameters (segment and offset of GPIB handler, and address of status words) are always required, so these are supplied by the program.

The high-level Forth word GPIB actually performs each interface bus function. The syntax is:

```
buf-adr cnt dev_descr ib_fcn gpib
or
value       dev_descr ib_fcn gpib
```

where:
value is any value needed by a function,
buf-adr is the buffer address,
cnt is the length of the buffer,
dev_descr is the device descriptor, and
ib_fcn is one of the constants given in the source code.

Note that buf-adr and cnt are required for functions such as reading, writing, and sending commands to a device. Other functions require that a different parameter replace buf-adr and cnt.

In the source code after GPIB are some words which get one character and get a string of characters from the GPIB device. Characters from the device are fetched one at a time and stored in PAD. They are concatenated into a working buffer called STRNGBUF.

The Philips frequency counter sends numbers in scientific notation, which is not valid and convertible by NUMBER. So $>D is used to change the letter "E" to a blank, then the string can be converted to a double by NUMBER. This, in effect, ignores the exponent.

The word called KLUDGE deserves special mention. In converting scientific notation to a number, the leading "+" in front of the exponent is not recognized by NUMBER. Generally, if we can ignore the exponent, the mantissa will be converted to a double. In cases where one wants to use software floating point, the + needs to be converted to a zero. The work KLUDGE does this. The hardware floating-point routines do not need this. Alternately, one could rework NUMBER to account for the leading plus sign. (Note: these are the software and hardware floating-point packages provided with F-PC from the Forth Interest Group. Both packages were written by Robert Smith.)

Words which set the various operating modes of the Philips frequency counter make up the next section of the source code. The counter can accept two inputs, and can be selected to read the frequency of a or b channels, revolutions per minute or period of a, totalize a or b, or the ratio of a to b. Other words open and close the gate for manual operation, set the measurement time and time-out values, set triggering modes, service request mask, output mode, and output separator. There are also words which query the counter to return the operating modes.

The word INITFREQ initializes the frequency counter with the default operating parameters. It will abort if IBFIND cannot find the device on the interface bus; otherwise, it will print the return code (or device descriptor) that was assigned to the frequency counter by the National Instruments handler.

The Forth word XTAL takes a filename, aborts if the file already exists, otherwise opens the file, then starts reading values from the timer/counter, printing elapsed time and the value both on the screen and to the given file in ASCII format. This file can later be imported into a spreadsheet program, with which a 2-D graph can be generated.

References

Philips Test and Measurement Department Inc., 12882 Valley View Street, Suite 9, Garden Grove, CA 92645.

National Instruments, 12109 Technology Boulevard, Austin, TX 78727-6204.

Forth Interest Group, P.O. Box 2154, Oakland, CA 94621.

Robert Thompson started programming in Fortran in 1976 in college, then later advanced to BASIC. By the time he graduated from college in 1980, he had mastered the art of punching Hollerith cards and using line numbers. Around 1982, he started programming in Turbo Pascal on Zenith and IBM microcomputers while working toward a master's degree in chemistry. A couple of years later, he was introduced to Forth by one of his college professors. Being of a curiously bent nature, he obtained a copy of F83 and began to undo all his laboriously learned thought processes. He now works as a chemist in an environmental testing laboratory, and programs in his spare time.

```
\ source code starts here
\ file gpibstuff.seq starts here
hex
\ these are the various functions that gpib can do...
\ they are defined as constants
\ hi level syntax for calling a function is:
\       parms dev-handle ibxxxx gpib
\
\ function constant ..... description ............ syntax for usage
\
0 constant IBWAIT \ wait for selected event:    mask dev ibwait gpib
1 constant IBONL \ place device online/offline: v dev ibonl gpib
2 constant IBRSC \ request/release system control: v dev ibrsc gpib
3 constant IBSIC \ send interface clear:        dev ibsic gpib
4 constant IBSRE \ set/clear remote enable line: v dev ibsre gpib
5 constant IBLOC \ go to local:                 dev ibloc gpib
6 constant IBRSV \ request service:             v dev ibrsv gpib
7 constant IBPPC \ parallel poll configure:     v dev ibppc gpib
8 constant IBPAD \ change primary address:      v dev ibpad gpib
9 constant IBSAD \ change secondary address:    v dev ibsad gpib
0a constant IBIST \ set/clear individual status bit: v dev ibist gpib
0b constant IBDMA \ enable/disable dma:         v dev ibdma gpib
0c constant IBEOS \ change/disable eos mode:    v dev ibeos gpib
0d constant IBTMO \ change/disable time limit:  v dev ibtmo gpib
0e constant IBEOT \ enable/disable end message: v dev ibeot gpib
0f constant IBGTS \ go from active controller to standby: v dev ibgts gpib
10 constant IBCAC \ become active controller:   v dev ibcac gpib
11 constant IBRDF \ read data to file:          fname-adr cnt dev ibrdf gpib
12 constant IBWRTF \ write data from file:      fname-adr cnt dev ibwrtf gpib
13 constant IBRPP \ conduct a parallel poll:    ppr-adr cnt dev ibrpp gpib
14 constant IBPOKE
15 constant IBSTOP \ abort asynchronous operation: dev ibstop gpib
16 constant IBCLR \ clear specified device:     dev ibclr gpib
17 constant IBTRG \ trigger selected device:    dev ibtrg gpib
18 constant IBPCT \ pass control:               dev ibpct gpib
19 constant IBRSP \ return serial poll byte:    spr-adr cnt dev ibrsp gpib
1c constant IBRD \ read data:                   rd-buf cnt dev ibrd gpib
1d constant IBRDA \ read data asynchronously:   rd-buf cnt dev ibrda gpib
1e constant IBWRT \ write data:                 wrt-buf cnt dev ibwrt gpib
1f constant IBWRTA \ write async:               wrt-buf cnt dev ibwrta gpib
20 constant IBCMD \ send commands:              cmd-buf cnt dev ibcmd gpib
21 constant IBCMDA \ send commands async:       cmd-buf cnt dev ibcmda gpib
22 constant IBDIAG
23 constant IBXTRC
24 constant IBTRAP

\ gpib commands
\ one of these values can replace v above
01 constant GTL \ goto local
04 constant SDC \ selected device clear
05 constant PPC \ parallel poll configure
08 constant GET \ group execute trigger
09 constant TCT \ take control
11 constant LLO \ local lock out
14 constant DCL \ device clear
15 constant PPU \ parallel poll unconfigure
18 constant SPE \ serial poll enable
19 constant SPD \ serial poll disable
3f constant UNL \ unlisten
5f constant UNT \ untalk
60 constant PPE \ parallel poll enable
70 constant PPD \ parrallel poll disable

\ gpib status bit vector and mask values for ibwait
```

```
8000 constant ERR \ error detected
4000 constant TIMO \ timeout
2000 constant UEND \ eoi or eos detected
1000 constant SRQI \ srq detected by cic
 800 constant RQS \ device needs service
 100 constant CMPL \ i/o completed
  80 constant LOK \ local lockout state
  40 constant REM \ remote state
  20 constant CIC \ controller-in-charge
  10 constant ATN \ attention asserted
   8 constant TACS \ talker active
   4 constant LACS \ listener active
   2 constant DTAS \ device trigger state
   1 constant DCAS \ device clear state
decimal

\ values to use for time out
 0 constant 0timeout
 1 constant 10usec
 2 constant 30usec
 3 constant 100usec
 4 constant 300usec
 5 constant 1msec
 6 constant 3msec
 7 constant 10msec
 8 constant 30msec
 9 constant 100msec
10 constant 300msec
11 constant 1sec
12 constant 3sec
13 constant 10sec
14 constant 30sec
15 constant 100sec
16 constant 300sec
17 constant 1000sec

comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        the first three words in GPIBBUF are set by IBFIND (really by DOS
I/O control and the National Instruments device driver).  IBFIND clears
the next three words.  They will be filled by the device driver after
each gpib function call.  The device driver expects these variables to be
contiguous in memory.

        diagram of gpib buffer:
        offset    segment   handle    ibstatus  iberror   ibcount
        [2 bytes] [2 bytes] [2 bytes] [2 bytes] [2 bytes] [2 bytes]
          |         |         |           |         |         |
          |         |         |           V         V         V
          |         |         V         set by driver with each call
          V         V    handle of device returned from driver
         address of device driver
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

12 constant b/gpibbuf
create gpibbuf b/gpibbuf allot
gpibbuf b/gpibbuf erase     \ zero buffer

\ These names return the address so they can be accessed
\ just like regular variables
: ibsta ( -- adr ) gpibbuf 6 + ;        \ status word
: iberr ( -- adr ) gpibbuf 8 + ;        \ gpib error code
: ibcnt ( -- adr ) gpibbuf 10 + ;       \ number of bytes sent or dos error
```

```
: .vars ( - ) \ print interface bus variables
        ." ibsta=" ibsta @ .
        ."  iberr=" iberr @ .
        ."  ibcnt=" ibcnt @ . cr ;

: .ibsta ( - )
        cr ." ib status: " cr
        ."  etesr clrcatldd" cr \ read these mnemonics vertically
        ."  rinrq moeitaatc" cr
        ."  rmdqs pkmcnccaa" cr
        ."   o i  l    ssss" cr
        base @ ibsta @ 2 base ! [ decimal ]
        dup 0>= if 16 else 15 then .r base !
        cr ;

: .iberr ( - )
        ." ib error: "
        iberr @
        dup  0=  if ." no error or dos error" then
        dup  1 = if ." gpib-pc must be cic" then
        dup  2 = if ." no listener on write function" then
        dup  3 = if ." gpib-pc not addressed correctly" then
        dup  4 = if ." bad arg to function call" then
        dup  5 = if ." gpib-pc not system controller " then
        dup  6 = if ." i/o aborted" then
        dup  7 = if ." no gpib-pc board" then
        dup 10 = if ." i/o started before prev op compl" then
        dup 11 = if ." no capability for operation" then
        dup 12 = if ." file system error" then
        dup 14 = if ." command error during device call" then
        dup 15 = if ." serial poll status byte lost" then
        dup 16 = if ." srq stuck in on position" then
        drop cr ;


comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        This routine was written to call the National Instruments device
driver, and ask it to address a device on the GPIB.
        We use DOS I/O control to get addr of the device driver, and pass
it the handle of the device we want to access.  adr-buf is a buffer that
is filled with information, according to the above table, by DOS and the
device driver.  It is information that will be needed for accessing a
device.  A buffer must be allocated and a handle put on stack prior to
calling this function.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
comment;

hex
code ioctlread ( nbytes adr-buf handle -- nbytes|errcode carry )
        pop bx  \ handle
        mov ax, # 4402 \ ioctl, subfn read
        pop dx  \ adr of data area
        pop cx  \ nbytes, value doesn't matter
        int 21
        push ax \ push nbytes or error code
        lahf \ get the flags
        mov al, ah
        xor ah, ah
        and ax, # 1 \ isolate the carry bit, if set then error
        1push
        end-code
decimal
```

```
\ dev10 is the name by which the National Instruments device driver
\ will refer to device number 10 (our frequency counter) on the gpib.
\ Other devices have similar monikers.
handle pm6680handle     pm6680handle !hcb dev10
handle pm6665handle     pm6665handle !hcb dev11 \ our old freq counter
handle gpib0handle      gpib0handle !hcb gpib0
\ for example: if you had a plotter as device # 6 then use...
\ handle plotterhandle  plotterhandle !hcb dev06

\ The following handles should be different for each device on the gpib.
0 value pm6680  \ gpib handle number, value returned from ibfind
0 value pm6665
\ Its possible to have only two boards in one computer.
0 value gpib0   \ two boards are named gpib0 and gpib1

: ibfind ( hcb -- handle# )
\ IBFIND finds the board or device from the installed gpib driver
\ the board and/or device names are in handle control blocks.
\ IBFIND returns the board or device number, which is -1 if error
\ board or device numbers should be put in global vars for later use.
        read-only def-rwmode    \ set read only mode
        dup>R hopen
        0= if
                0 gpibbuf R@ >hndle @ ioctlread
\                not if abort"  error in ioctlread " then drop
                2drop
                gpibbuf 4 + @ dup  \ put handle on stack
                0<= if
                        drop -1
                then
                R@ hclose drop
        then R> drop
        ;

: getdriver ( -- seg ofs )
\ returns seg,ofs of device driver from global buffer GPIBBUF
        gpibbuf 2+ @    \ segment
        gpibbuf @ ;     \ offset


comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        This routine was written to pass parameters to the National
Instruments device driver.
        All registers that FPC uses must be saved around this routine,
these are: bp, ds, es, and si
        CALLGPIB calls a TSR program outside of FPC, passing some parms
to it; then returns to FPC.  The first two parms (adr-buf, cnt) will vary
depending on the function call, other parms are needed in any case.  The
invariant parms (segment and offset of device driver, and ib status word)
are pushed by the forth word GPIB; others are the responsibility of
the user.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
comment;

code callgpib ( adr-buf cnt handle function seg ofs ibsta -- )
\               14      12      10      8       6   4   2    <-offset from bp
        push bp \ fpc uses bp as ret stk pointer
        mov bp, sp
        push ds
        push es \ driver uses es as segment to store ibsta, iberr, ibcnt
        push si
        mov ax, 8 [bp] \ function in ax
        push ds
```

```
        pop es \ make sure es points to data segment for driver
        mov si, 2 [bp] \ si points to ib vars
        mov bx, 10 [bp] \ device handle in bx
        mov dx, 14 [bp] \ buf-adr in dx
        mov cx, 12 [bp] \ count in cx
        call far [] 4 [bp] \ call driver
        pop si \ restore regs ...
        pop es
        pop ds
        pop bp
        next
        end-code

comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
hi-level syntax for gpib functions will be:

        parms dev-handle ibxxxx gpib

parms is whatever the particular function requires
handle is handle of device to address.
ibxxxx is one of the function name constants
On return from this call, we drop the junk left on the stack.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

: gpib
getdriver ibsta callgpib 3drop
        EXEC:   \ numbers in hex
\        0       1       2       3       4       5       6       7
        2drop   2drop   2drop   drop    2drop   drop    2drop   2drop
\        8       9       0a      0b      0c      0d      0e      0f
        2drop   2drop   2drop   2drop   2drop   2drop   2drop   2drop
\        10      11      12      13      14      15      16      17
        2drop   3drop   3drop   3drop   2drop   drop    drop    drop
\        18      19      1a      1b      1c      1d      1e      1f
        drop    3drop   noop    noop    3drop   3drop   3drop   3drop
\        20      21      22      23      24
        3drop   3drop   3drop   3drop   2drop
        ;

0 value dev
\ use the constant DEV to store current device handle
\ this will make it easier to send commands to device while interpreting.
\ make the device current by typing: pm6665 >dev
\ then all commands sent with IB" will go to this device
: >dev ( handle -- ) =: dev ;

comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        We can send a command direct to the frequency counter by
typing:  ib" <command>" .  Be careful NOT to use this when
collecting data in background with the multitasker.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

\ ib" sends a string to a gpib device in interpretive state
\ use >ib in compiling state
\ examples:
\ first set desired device with >DEV
\  pm6680 >dev
\  ib" *rst"          \ resets to default state
\  ib" :acq:aper 1"      \ sets count time to 1 second
: ib" ( | string" -- ) \ for interpreting, print to gpib
        ascii " parse dev ibwrt gpib ;
```

```
: >ib ( adr len -- ) \ for compiling, print to gpib
    dev ibwrt gpib ;

255 constant len-of-buf
create strngbuf  len-of-buf 1+ allot \ 1 bytes for length
strngbuf len-of-buf blank

\ fetch one character from a device
: ib-1in ( -- char ) pad 1 dev ibrd gpib pad C@ ;

: ib-in ( -- adr )
\ fetch a line of input from a device
\ we could build on IB-1IN and get one char at a time until
\ no more are available, but this way is faster.
\ NOTE: Be sure that EOI is enabled
        strngbuf 1+ len-of-buf dev ibrd gpib \ get line of input from gpib
        ibcnt @ 1- strngbuf c! \ store length byte less lf at end
        strngbuf \ return address on stack
        ;

\ find out if pm6680 device is connected and healthy
\ method: try to obtain the status byte, then check for timeout.
\ if timeout received, then device is turned off, return false flag
: awake? ( dev -- flag )
    pad 1 rot ibrsp gpib
    ibsta @ TIMO and 0= if true else false then
    ;

\ file strings.seq starts here
\ words to manipulate strings
\ add two strings together (concatenate)
\ string2 is added to the end of string1
254 constant buflen
create tempbuf buflen 1+ allot
tempbuf buflen erase
: $+ ( string1 string2 -- string1+string2 ) \ string is adr len
\        dup>R pad 1+ swap cmove pad 1+ R@ + swap dup>R cmove pad
\        R> R> + swap c! pad count ;
    dup>R 2swap dup>R 2dup tempbuf 1+ swap cmove tempbuf + 1+
    nip swap cmove R> R> + tempbuf c!
    tempbuf count ;

\ make a single number into a string
: n>$ ( n1 -- adr len ) S>D <# #S #> ;

\ make a double number into a string
: D>$ ( d -- adr len ) <# #S #> ;

: F>$ ( F: r -- ; -- adr cnt ) \ float to string
\ : F.     ( F: r -- )
        F#PLACES @ 1 MAX 10 MIN
        FDUP0=
        IF      FDROP F0.0  THEN
        [ ALSO HIDDEN ] (F.1)
        IF      DROP FNIP [ LAST @ NAME> ] LITERAL 64 FPERR
                IF   FNEGATE  THEN 10 (E.)
        ELSE    FSWAP (F.FRACT) (F.INT1)
        THEN    \ TYPE SPACE
    ;  PREVIOUS

: dash 196 emit ;
\ draw horz line n chars long
: dashes ( n -- ) 0 do dash loop ;
```

```
\ return leftmost chars in a string
: left$ ( adr len cnt -- adr' len' ) min ;

\ return rightmost chars in a string
: right$ ( adr len cnt -- adr' len' )
        dup>R - + R> ;

\ return substring of a string
: mid$ ( adr len index cnt -- adr' len' )
        >R rot + 1- swap R> min ;

\ print string center justified
: $.C ( adr len field-len -- ) 2dup >R >r
        <= if R> R> over - 2/ dup>R spaces type R> spaces
        else R> R@ tuck - 2/ R> mid$ type
        then ;
\                 1         2         3         4
\        12345678901234567890123456789012345678901
\ : test " 0123456789this is a test string9876543210" ;

\ : s1 " ...This is the first string..." ;
\ : s2 " ...The second string is this one. How about that?..." ;

\ remove leading spaces and nulls from a string
: -leading ( adr len -- adr' len' )
    begin
        over c@ dup 0= swap bl = or while
            1- swap 1+ swap
    repeat
    ;


\ file timestuf.seq starts here
: get-time ( -- dmin )
\ returns time in minutes
        gettime
        >R dup $FF and >R 256 / R>
        R> dup $FF and >R 256 / R>
        3 roll 60 *
        3 roll + 10000 um*
        3 roll 100 *
        3 roll +
        100 60 */
        S>D D+
        ;

\ turns my time into a string
: time>$ ( dmin n -- adr len )
        <# 0 do # loop ascii . hold #S #> ;

: .time ( -- ) \ prints time in minutes
        get-time time>$ type ;


\ file pm6680.seq starts here
\ anew xtalfile

\ !used

\ fload gpibstuf.seq
fload multask.seq  \ from f-pc
\ fload strings.seq
\ fload timestuf.seq
```

```
comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        The next word, DATAREADY?, will examine the status byte which is
returned from the frequency counter when the gpib is told to poll the
device and fetch it.  We could fetch this byte by directly sending
commands to the counter, but it makes better sense to let the gpib do this
work.  After all, that's what we have it for!  A flag is left on the stack
when the counter has asserted the request service line of the gpib (due to
there being a message available if 16; or due to the enabled operation
having occurred if 128).
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

variable spb \ serial poll byte
: getsb ( -- n ) \ return the status byte from pm6680
    spb 1 pm6680 ibrsp gpib spb @ ;

: dataready?  ( -- flag )
        getsb 128 and 0> if true else false then \ enab op.stat occurred
        ;
: msgavail? ( -- flag )
        getsb 16 and 0> if true else false then \ mes.avail
        ;
: error? ( -- flag )
        getsb 4 and 0> if true else false then \ err.avail
        ;

\ function selecting commands
: local pm6680 ibloc gpib ;
: reset " *rst" pm6680 ibwrt gpib  local ; \ resets counter to default values
: clear " *cls" pm6680 ibwrt gpib  local ; \ clears command and error queues
: freq-a " :func 'freq 1'" pm6680 ibwrt gpib  local ;
: freq-b " :func 'freq 2'" pm6680 ibwrt gpib  local ;
: per-a  " :func 'per 1'" pm6680 ibwrt gpib  local ;
: tota-b " :func 'tot 1,2'" pm6680 ibwrt gpib  local ;
: ratio-ab " :func 'freq:rat 1,2'" pm6680 ibwrt gpib  local ;

\ input setting commands
: trig-pos " :inp:slop pos" pm6680 ibwrt gpib  local ;
: trig-neg " :inp:slop neg" pm6680 ibwrt gpib  local ;

\ measurement control commands
: mtime ( hund-sec -- ) \ set measuring time
        " :acq:aper " rot S>D <# # # ascii . hold #S #>
        $+ pm6680 ibwrt gpib
        ;

comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        In the following initialization of the counter, we tell it to
let us know when a measurement has stopped.  This way we can monitor the
status of the counter, and while waiting for it to finish a measurement,
we can tell Forth to do something else.  This is necessary for multitasking.
        The statement ":stat:oper:enab 256" says "enable the change to the
operation status when a measurement has stopped".  (256=wait for meas stop;
64=wait for bus arming; 32=wait for trigger; 16=wait for meas start).
Add these values together to wait for a combined event.
        The statement "*sre 132" says "enable the status request line when
the enabled operation has occurred (128) and event status bit (32)". We could
also say "*sre 20" which says "enable the status request line when a
message is available (16) or error ready (4)".  Both of these appear to
work equally well.
        The statement "*ese 36" is command error=32, execution error=16,
device dependent error=8, query error=4.
        These statements must be used to properly enable
```

```
the counter to assert the request service line of the gpib so we
can do data collection in background.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

\ meas.times given in hundredths of seconds
100 constant meas-time \ 1 second

\ the following are bit masks for status events
\ status operation enable
256 constant meas.stop
64 constant arm.wait
32 constant trig.wait
16 constant meas.start

\ standard event status enable          service request enable
128 constant power.on                   128 constant op.stat
64 constant user.req                    64 constant req.serv
32 constant com.err                     32 constant ev.stat
16 constant exec.err                    16 constant mes.avail
8 constant dev.err                      8 constant ques.sig
4 constant query.err                    4 constant err.avail
2 constant req.contr                    1 constant dev.stat
1 constant op.compl

\ if error, then print status and error messages and abort
\ else set status requests, meas.time, and return counter to local mode
: initfreq ( - )
    gpib0handle ibfind dup ." board handle=" . =: gpib0
    pm6680handle ibfind dup ." , device handle=" . =: pm6680 cr
    gpib0 pm6680 or 0< iberr @ or abort" abort on ibfind"
    reset
    " :stat:oper:enab " meas.stop N>$ $+ pm6680 ibwrt gpib
    " *ese " com.err query.err + op.compl + N>$
        $+ pm6680 ibwrt gpib
    " *sre " op.stat req.serv + ev.stat + dev.stat + N>$
        $+ pm6680 ibwrt gpib
    meas-time mtime local
    pm6680 >dev \ make this device current
\     ." Be sure freq counter is turned on, or FPC will hang. "
    cr
    ;

comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        After receiving a string of characters from the frequency counter,
which represent a frequency (or other) reading, because of some limitations
in Forth, we have to fudge some characters in the string.  Particularly
annoying, is the leading plus sign, on which NUMBER will (but shouldn't)
abort.  So, we change it to a zero.  We also have to rid ourselves of
the E before the exponent.  Change it to a blank and NUMBER will
ignore the exponent.  Simple!  Forth doesn't know that the number from
the counter is a float anyway, it is treated as a double. (There is also
a limit on how many digits in the number Forth will convert
to a double.)  Too many and the value that can be contained in a double
overflows.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

\ put a zero at plus sign in front of mantissa. number needs this.
\ put a space at position of exponent. number needs this.
\ ignore the exponent.
\ could use hardware floating point version of number,
\ it reads number with leading '+'.  software version does not.
```

```
\ From the pm6680, the string of characters is always the same length.
: freq$>d ( adr - d )
        dup 1+ '0' swap c! \ replace + with 0
        dup 12 + bl swap c! \ put space at E
        number ;

\ gets freq from counter as double.
\ multitasking allowed.  assumes that stat:operation:enable is true
\ when measurements started.  serial polls the rqs line, and pauses
\ until line is true (or measurement has stopped).
: getdata ( -- d )  " :read?" pm6680 ibwrt gpib
\ get one reading and return as double value on stack
        begin msgavail? not while pause repeat
        ib-in freq$>d
        ;

handle xtalhandle
create crlf$ $0D c, $0A c,

comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        COLLECT is a background task that will fetch frequency data from
the counter and store time and frequency readings into a disk file.  Once
it is running, it can be interrupted by typing SINGLE and restarted by
typing MULTI, but settings on the frequency counter cannot be changed
in the between time.  This will cause Forth to crash.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

variable datacnt \ number of this data point
2variable starttime
2variable currtime \ current time
\ 2variable freqdata
80 constant freqlen
create freqdata freqlen allot \ this holds the last point collected
freqdata freqlen erase
variable gooddata \ a flag that is set when datum is collected
\ gooddata is cleared when the data has been displayed on screen
gooddata off
\ write the elapsed time and freq to file
\ stop when a write error occurs
background: collect
    datacnt off cr
    begin
        incr> datacnt
        get-time starttime 2@ d- 2dup currtime 2!
        4 time>$ xtalhandle hwrite drop
        "  " xtalhandle hwrite drop
\        getdata 2dup freqdata 2!
        " read?" >ib begin msgavail? not while pause repeat ib-in
        dup c@ 1+ freqdata swap cmove
        gooddata on pause
        freqdata count xtalhandle hwrite drop
\        D>$ xtalhandle hwrite drop
        crlf$ 2 xtalhandle hwrite drop
    false until
    xtalhandle hclose drop local
    ;

comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        This TASK1 is an experiment in multitasking.  It simply picks
up data from memory and displays it in a cute window on the screen.  The
COLLECT task must be running and collecting data before TASK1 will do
anything.  The screen display can be turned off by typing: 'showdata off'
```

```
        A flag is used (the variable GOODDATA) to tell when a
datum should be displayed.  COLLECT will set GOODDATA to true after it has
collected one point, TASK1 will set GOODDATA to false after the number has
been displayed on the screen.  This prevents TASK1 from constantly
redisplaying the same number.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

variable showdata \ flag for screen display of data
showdata on  \ yes we will display data
\ task1 will get data from memory and print it on the screen
\ depending on if showdata is true.
\ this depends on collect to actually get data from the freq counter.
\ this runs in the background.
background: task1
    begin pause
        showdata @ if
            gooddata @ if
                savecursor 45 1 78 3 box&fill
                46 2 at datacnt @ . "  " type
                currtime 2@ 4 time>$ type "  " type
                freqdata count type
\                62 2 at freqdata 2@ 15 d.r
                gooddata off
                restcursor
            then
        then
    again ;

create errmsg 52 allot
errmsg 50 blank
variable diderror  diderror off \ did we already show this error?
: geterrormsg ( -- adr )
        " :syst:err?" pm6680 ibwrt gpib  ib-in
        diderror off
        ;

variable showerror \ flag for screen display of error msg
showerror on  \ yes we will display error msg
\ task2 will get error no. from freq counter and print msg on screen
\ depending on if showerror is true.
\ this runs in the background.
background: task2
    begin pause
        showerror @ if \ if we are to show this error
            error? if \ if an error has occurred
                savecursor
                geterrormsg count 78 over - 1- dup>R 1 78 3 box&fill
                R> 1+ 2 at type
                restcursor 10 ms \ restore cursor, and wait a short while
            then
        then
    again ;

variable showsb \ flag for screen display of status byte
showsb on  \ yes we will display status byte
\ task3 will get status byte from freq counter and print it on screen
\ depending on if showsb is true.
\ this runs in the background.
background: task3
    begin pause
        showsb @ if \ if we are to show this status byte
            savecursor 69 4 78 9 box&fill
```

FIG
MAIL ORDER FORM

HOW TO USE THIS FORM: Please enter your order on the back page of this form and send with your payment to the Forth Interest Group.
All items have one price and a weight marked with a # sign. Enter weight on order form and calculate shipping based on location and delivery method.

"Were Sure You Wanted To Know..."

***Forth Dimensions*, Article Reference** 151 - $4 0#
★ An index of Forth articles, by keyword, from *Forth Dimensions* Volumes 1–13 (1978–92).

FORML, Article Reference 152 - $4 0#
★ An index of Forth articles by keyword, author, and date from the FORML Conference Proceedings (1980–91).

FORTH DIMENSIONS BACK VOLUMES

A volume consists of the six issues from the volume year (May–April)

Volume 1 Forth Dimensions (1979–80) 101 - $15 1#
Last 50
Introduction to FIG, threaded code, TO variables, fig-Forth.

Volume 6 Forth Dimensions (1984–85) 106 - $15 2#
Interactive editors, anonymous variables, list handling, integer solutions, control structures, debugging techniques, recursion, semaphores, simple I/O words, Quicksort, high-level packet communications, China FORML.

Volume 7 Forth Dimensions (1985–86) 107 - $20 2#
Generic sort, Forth spreadsheet, control structures, pseudo-interrupts, number editing, Atari Forth, pretty printing, code modules, universal stack word, polynomial evaluation, F83 strings.

Volume 8 Forth Dimensions (1986–87) 108 - $20 2#
Interrupt-driven serial input, data-base functions, TI 99/4A, XMODEM, on-line documentation, dual CFAs, random numbers, arrays, file query, Batcher's sort, screenless Forth, classes in Forth, Bresenham line-drawing algorithm, unsigned division, DOS file I/O.

Volume 9 Forth Dimensions (1987–88) 109 - $20 2#
Fractal landscapes, stack error checking, perpetual date routines, headless compiler, execution security, ANS-Forth meeting, computer-aided instruction, local variables, transcendental functions, education, relocatable Forth for 68000.

Volume 10 Forth Dimensions (1988–89) 110 - $20 2#
Last 50
dBase file access, string handling, local variables, data structures, object-oriented Forth, linear automata, stand-alone applications, 8250 drivers, serial data compression.

Volume 11 Forth Dimensions (1989–90) 111 - $20 2#
Local variables, graphic filling algorithms, 80286 extended memory, expert systems, quaternion rotation calculation, multiprocessor Forth, double-entry bookkeeping, binary table search, phase-angle differential analyzer, sort contest.

Volume 12 Forth Dimensions (1990–91) 112 - $20 2#
Floored division, stack variables, embedded control, Atari Forth, optimizing compiler, dynamic memory allocation, smart RAM, extended-precision math, interrupt handling, neural nets, Soviet

FORML CONFERENCE PROCEEDINGS

FORML (Forth Modification Laboratory) is an educational forum for sharing and discussing new or unproven proposals intended to benefit Forth, and is an educational forum for discussion of the technical aspects of applications in Forth. Proceedings are a compilation of the papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group.

1980 FORML PROCEEDINGS 310 - $30 2#
Last 10
Address binding, dynamic memory allocation, local variables, concurrency, binary absolute & relocatable loader, LISP, how to manage Forth projects, n-level file system, documenting Forth, Forth structures, Forth strings. *231 pgs*

1981 FORML PROCEEDINGS 311 - $45 4#
Last 50
CODE-less Forth machine, quadruple-precision arithmetic, overlays, executable vocabulary stack, data typing in Forth, vectored data structures, using Forth in a classroom, pyramid files, BASIC, LOGO, automatic cueing language for multimedia, NEXOS—a ROM-based multitasking operating system. *655 pgs*

1982 FORML PROCEEDINGS 312 - $30 4#
Rockwell Forth processor, virtual execution, 32-bit Forth, ONLY for vocabularies, non-IMMEDIATE looping words, number-input wordset, I/O vectoring, recursive data structures, programmable-logic compiler. *295 pgs*

1983 FORML PROCEEDINGS 313 - $30 2#
Non-Von Neuman machines, Forth instruction set, Chinese Forth, F83, compiler & interpreter co-routines, log & exponential function, rational arithmetic, transcendental functions in variable-precision Forth, portable file-system interface, Forth coding conventions, expert systems. *352 pgs*

1984 FORML PROCEEDINGS 314 - $30 2#
Forth expert systems, consequent-reasoning inference engine, Zen floating point, portable graphics wordset, 32-bit Forth, HP71B Forth, NEON—object-oriented programming, decompiler design, arrays and stack variables. *378 pgs*

1986 FORML PROCEEDINGS 316 - $30 2#
Threading techniques, Prolog, VLSI Forth microprocessor, natural-language interface, expert system shell, inference engine, multiple-inheritance system, automatic programming environment. *323 pgs*

1987 FORML PROCEEDINGS 317 - $40 3#
Last 25
Includes papers from '87 euroFORML Conference. 32-bit Forth, neural networks, control structures, AI, optimizing compilers, hypertext, field and record structures, CAD command language, object-oriented lists, trainable neural nets, expert systems. *463 pgs*

1988 FORML PROCEEDINGS 318 - $40 2#
Includes 1988 Australian FORML, Human interfaces, simple robotics kernel, MODUL Forth, parallel processing, programmable controllers, Prolog, simulations, language topics, hardware, Wil's workings & Ting's philosophy, Forth hardware applications, ANS Forth session, future of Forth in AI applications. *310 pgs*

1989 FORML PROCEEDINGS 319 - $40 3#
Last 50
Includes papers from '89 euroFORML. Pascal to Forth, extensible optimizer for compiling, 3D measurement with object-oriented Forth, CRC polynomials, F-PC, Harris C cross-compiler, modular approach to robotic control, RTX recompiler for on-line maintenance, modules, trainable neural nets. *433 pgs*

1990 FORML PROCEEDINGS 320 - $40 3#
Last 10
Forth in industry, communications monitor, 6805 development, 3-key keyboard, documentation techniques, object-oriented programming, simplest Forth decompiler, error recovery, stack operations, process control event management, control structure analysis, systems design course, group theory using Forth. *441 pgs*

1991 FORML PROCEEDINGS 321 - $50 3#
Includes 1991 FORML (Asilomar), euroFORML '91 (Czechoslovakia) and 1991 China FORML (Shanghai). differential file comparison, LINDA on a simulated network, QS2: RISCing it all, A threaded microprogram machine, Forth in networking, Forth in the Soviet Union, FOSM: A FOrth String Matcher, VGA Graphics and 3-D animation, Forth and TSR, Forth CAE system, applying Forth to electric discharge machining, MCS96-FORTH single chip computer. *500 pgs*

★ These are your most up-to-date indexes for back issues of *Forth Dimensions* and the FORML proceedings.

Fax your orders: 510-535-1295

BOOKS ABOUT FORTH

ALL ABOUT FORTH, 3rd ed., June 1990, Glen B. Haydon 201 - $90 4#
Annotated glossary of most Forth words in common usage, including Forth-79, Forth-83, F-PC, MVP-Forth. Implementation examples in high-level Forth and/or 8086/88 assembler. Useful commentary given for each entry. *504 pgs*

THE COMPLETE FORTH, Alan Winfield 210 - $14 1#
A comprehensive introduction, including problems with answers (Forth-79). *131 pgs*

eFORTH IMPLEMENTATION GUIDE, C.H. Ting 215 - $25 1#
eForth is the name of a Forth model designed to be portable to a large number of the newer, more powerful processors available now and becoming available in the near future. *54 pgs* (w/disk)

Embedded Controller FORTH, 8051, William H. Payne 216 - $65 2#
Now Available
Describes the implementation of an 8051 version of Forth. More than half of this book contains source listings (See disks C050).. Made available at the request of members. *511 pgs*

F83 SOURCE, Henry Laxen & Michael Perry 217 - $20 2#
A complete listing of F83, including source and shadow screens. Includes introduction on getting started. *208 pgs*

FORTH: A TEXT AND REFERENCE 219 - $31 2#
Mahlon G. Kelly & Nicholas Spies
A textbook approach to Forth, with comprehensive references to MMS-FORTH and the '79 and '83 Forth standards. *487 pgs*

THE FIRST COURSE, C.H. Ting 223 - $25 1#
This tutorial's goal is to expose you to the very minimum set of Forth instructions you need to use Forth to solve practical problems in the shortest possible time. "... This tutorial was developed to complement *The Forth Course* which skims too fast on the elementary Forth instructions and dives too quickly in the advanced topics in a upper level college microcomputer laboratory ..." A running F-PC Forth system would be very useful. *44 pgs*

THE FORTH COURSE, Richard E. Haskell 225 - $25 1#
This set of 11 lessons, called *The Forth Course*, is designed to make it easy for you to learn Forth. The material was developed over several years of teaching Forth as part of a senior/graduate course in design of embedded software computer systems at Oakland University in Rochester, Michigan. *156 pgs* (w/disk)

FORTH ENCYCLOPEDIA, Mitch Derick & Linda Baker 220 - $30 2#
A detailed look at each fig-Forth instruction. *327 pgs*

FORTH NOTEBOOK, Dr. C.H. Ting 232 - $25 2#
Good examples and applications. Great learning aid. poly-FORTH is the dialect used. Some conversion advice is included. Code is well documented. 286 pgs

FORTH NOTEBOOK II, Dr. C.H. Ting 232a - $25 2#
Collection of research papers on various topics, such as image processing, parallel processing, and miscellaneous applications. *237 pgs*

FORTH: The New Model, Jack Woehr 233 - $45 2#
NEW
This book teaches Forth and the proposed new standard from the perspective of a Technical Committee member. You will find it especially helpful if you are: An experienced Forth programmer who wishes to become familiar with the draft-proposed standard for Forth, a Forth programmer who needs to know how to convert existing programs to the new proposed standard, a programmer, experienced in other languages, who is using Forth for an embedded control project, or a beginning Forth programmer who wishes to learn the language. *315 pgs, w/disk*

F-PC USERS MANUAL (2nd ed., V3.5) 350 - $20 1#
Users manual to the public-domain Forth system optimized for IBM PC/XT/AT computers. A fat, fast system with many tools. *143 pgs*

F-PC TECHNICAL REFERENCE MANUAL 351 - $30 2#
A must if you need to know the inner workings of F-PC. *269 pgs*

INSIDE F-83, Dr. C.H. Ting 235 - $25 2#
Invaluable for those using F-83. *226 pgs*

LIBRARY OF FORTH ROUTINES AND UTILITIES,
James D. Terry 237 - $23 2#
Comprehensive collection of professional quality computer code for Forth; offers routines that can be put to use in almost any Forth application, including expert systems and natural-language interfaces. *374 pgs*

OBJECT-ORIENTED FORTH, Dick Pountain 242 - $35 1#
Implementation of data structures. First book to make object-oriented programming available to users of even very small home computers. *118 pgs*

SEEING FORTH, Jack Woehr 243 - $25 1#
"... I would like to share a few observations on Forth and computer science. That is the purpose of this monograph. It is offered in the hope that it will broaden slightly the streams of Forth literature ..." *95 pgs*

SCIENTIFIC FORTH, Julian V. Noble 250 - $50 2#
Scientific Forth extends the Forth kernel in the direction of scientific problem solving. It illustrates advanced Forth programming techniques with non-trivial applications: computer algebra, roots of equations, differential equations, function minimization, functional representation of data (FFT, polynomials), linear equations and matrices, numerical integration/Monte Carlo methods, high-speed real and complex floating-point arithmetic. *300 pgs* (Includes disk with programs and several utilities), **IBM**

STACK COMPUTERS, THE NEW WAVE 244 - $62 2#
Philip J. Koopman, Jr. (hardcover only)
Presents an alternative to Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC) by showing the strengths and weaknesses of stack machines.

STARTING FORTH (2nd ed.), Leo Brodie 245 - $29 2#
In this edition of *Starting Forth*—the most popular and complete introduction to Forth—syntax has been expanded to include the Forth-83 Standard. *346 pgs*

WRITE YOUR OWN PROGRAMMING LANGUAGE USING C++,
Norman Smith 270 - $15 1#
This book is about an application language. More specifically, it is about how to write your own custom application language. The book contains the tools necessary to begin the process and a

ACM - SIGFORTH

The ACM SIGForth Newsletter is published quarterly by the Association of Computing Machinery, Inc. SIGForth's focus is on the development and refinement of concepts, methods, and techniques needed by Forth professionals.

Volume 1 Spring 1989, Summer 1989, #3, #4 911 - $24 2#
F-PC, glossary utility, euroForth, SIGForth '89 Workshop summary (real-time software engineering), Intel 80x8x. Metacompiler in cmForth, Forth exception handler, string case statement for UF/Forth. 1802 simulator, tutorial on multiple threaded vocabularies. Stack frames, duals: an alternative to variables, PocketForth.

Volume 2 #1, #2, #3, #4 912 - $24 2#
ACM SIGForth Industry Survey, abstracts 1990 Rochester conf., RTX-2000. BNF Parser, abstracts 1990 Rochester conf., F-PC Teach. Tethered Forth model, abstracts 1990 SIGForth conf. Target-meta-cross-: an engineer's viewpoint, single-instruction computer.

Volume 3, #1 Summer '91 913a - $6 1#
Co-routines and recursion for tree balancing, convenient number handling.

Volume 3, #2 Fall '91 913b - $6 1#
Postscript Issue, What is Postscript?, Forth in Postscript, Review: PS-Tutor.

1989 SIGForth Workshop Proceedings 931 - $20 1#
Software engineering, multitasking, interrupt-driven systems, object-oriented Forth, error recovery and control, virtual memory support, signal processing. *127 pgs*

1990-91 SIGForth Workshop Proceedings 932 - $20 1#
Teaching computer algebra, stack-based hardware, reconfigurable processors, real-time operating systems, embedded control, marketing Forth, development systems, in-flight monitoring, multi-processors, neural nets, security control, user interface, algorithms. *134 pgs*

For faster service, fax your orders: 510-535-1295

DISKS: Contributions from the Forth Community

The "Contributions from the Forth Community" disk library contains author-submitted donations, generally including source, for a variety of computers & disk formats. Each file is determined by the author as public domain, shareware, or use with some restrictions. This library does not contain "For Sale" applications. *To submit your own contributions, send them to the FIG Publications Committee.*

Count any number of disks as equal to 1 #.

FLOAT4th.BLK V1.4 Robert L. Smith C001 - $8
Software floating-point for fig-, poly-, 79-Std., 83-Std. Forths. IEEE short 32-bit, four standard functions, square root and log.
★★★ **IBM, 190Kb, F83**

Games in Forth C002 - $6
Misc. games, Go, TETRA, Life... Source.
★ **IBM, 760Kb**

A Forth Spreadsheet, Craig Lindley C003 - $6
This model spreadsheet first appeared in *Forth Dimensions* VII/1,2. Those issues contain docs & source.
★ **IBM, 100Kb**

Automatic Structure Charts, Kim Harris C004 - $8
Tools for analysis of large Forth programs, first presented at FORML conference. Full source; docs incl. in 1985 FORML Proceedings.
★★ **IBM, 114Kb**

A Simple Inference Engine, Martin Tracy C005 - $8
Based on inf. engine in Winston & Horn's book on LISP, takes you from pattern variables to complete unification algorithm, with running commentary on Forth philosophy & style. Incl. source.
★★ **IBM, 162 Kb**

The Math Box, Nathaniel Grossman C006 - $10
Routines by foremost math author in Forth. Extended double-precision arithmetic, complete 32-bit fixed-point math, & auto-ranging text. Incl. graphics. Utilities for rapid polynomial evaluation, continued fractions & Monte Carlo factorization. Incl. source & docs.
★★ **IBM, 118 Kb**

AstroForth & AstroOKO Demos, I.R. Agumirsian C007 - $6
AstroForth is the 83-Std. Russian version of Forth. Incl. window interface, full-screen editor, dynamic assembler & a great demo. AstroOKO, an astronavigation system in AstroForth, calculates sky position of several objects from different earth positions. Demos only.
★ **IBM, 700 Kb**

Forth List Handler, Martin Tracy C008 - $8
List primitives extend Forth to provide a flexible, high-speed environment for AI. Incl. ELISA and Winston & Horn's micro-LISP as examples. Incl. source & docs.
★★ **IBM, 170 Kb**

8051 Embedded Forth, William Payne C050 - $20
8051 ROMmable Forth operating system. 8086-to-8051 target compiler. Incl. source. Docs are in the book *Embedded Controller Forth for the 8051 Family*.
★★★ **IBM HD, 4.3 Mb**

68HC11 Collection C060 - $16
Collection of Forths, tools and floating point routines for the 68HC11 controller.
★★★ **IBM, 2.5 Mb**

F83 V2.01, Mike Perry & Henry Laxen C100 - $20
The newest version, ported to a variety of machines. Editor, assembler, decompiler, metacompiler. Source and shadow screens. Manual available separately (items 217 & 235). Base for other F83 applications.
★ **IBM, 83, 490 Kb**

F-PC V3.56 & **TCOM**, Tom Zimmer C200 - $30
A full Forth system with pull-down menus, sequential files, editor, forward assembler, metacompiler, floating point. Complete source and help files. Manual for V3.5 available separately (items 350 & 351). Base for other F-PC applications.
★ **IBM, 83, 3.5Mb**

F-PC TEACH V3.5, Lessons 0–7 Jack Brown C201 - $8
Forth classroom on disk. First seven lessons on learning Forth, from Jack Brown of B.C. Institute of Technology.
★ **IBM, F-PC, 790 Kb**

VP-Planner Float for F-PC, V1.01 Jack Brown C202 - $8
Software floating-point engine behind the VP-Planner spreadsheet. 80-bit (temporary-real) routines with transcendental functions, number I/O support, vectors to support numeric co-processor overlay & user NAN checking.
★★ **IBM, F-PC, 350 Kb**

F-PC Graphics V4.6, Mark Smiley C203 - $10
The latest versions of new graphics routines, including CGA, EGA, and VGA suppport, with numerous improvements over earlier versions created or supported by Mark Smiley.
★★ **IBM DSDD, F-PC, 605 Kb**

PocketForth V6.1, Chris Heilman C300 - $12
Smallest complete Forth for the Mac. Access to all Mac functions, events, files, graphics, floating point, macros, create standalone applications and DAs. Based on fig & *Starting Forth*. Incl. source and manual.
★ **MAC, 640 Kb,** System 7.01 Compatible.

Kevo V0.9b5, Antero Taivalsaari C360 - $10
Complete Forth-like object Forth for the Mac. Object-Prototype access to all Mac functions, files, graphics, floating point, macros, create standalone applications. Kernel source not included, extensive demo files, manual.
NEW VERSION
★★★ **MAC, 650 Kb,** System 7.01 Compatible.

Yerkes Forth V3.6 C350 - $20
Complete object-oriented Forth for the Mac. Object access to all Mac functions, files, graphics, floating point, macros, create standalone applications. Incl. source, tutorial, assembler & manual.
★★ **MAC, 2.4Mb,** System 7.01 Compatible.

Pygmy V1.4, Frank Sergeant C500 - $20
A lean, fast Forth with full source code. Incl. full-screen editor, assembler and metacompiler. Up to 15 files open at a time.
★★ **IBM, 320 Kb**

KForth, Guy Kelly C600 - $20
A full Forth system with windows, mouse, drawing and modem packages. Incl. source & docs.
★★ **IBM, 83, 2.5 Mb**

Mops V2.3, Michael Hore C710 - $20
Close cousin to Yerkes and Neon. Very fast, compiles subroutine-threaded & native code. Object oriented. Uses F-P co-processor if present. Full access to Mac toolbox & system. Supports System 7 (e.g., AppleEvents). Incl. assembler, manual & source.
NEW VERSION
★★ **MAC, 3 Mb,** System 7.01 Compatible

BBL & Abundance, Roedy Green C800 - $30
BBL public-domain, 32-bit Forth with extensive support of DOS, meticulously optimized for execution speed. Abundance is a public-domain database language written in BBL. Incl. source & docs.

New Version-Replacement Policy

Return the old version with the FIG labels and get a new version replacement for 1/2 the current version price.

MISCELLANEOUS

T-SHIRT "May the Forth Be With You" 601 - $12 1#
(Specify size: Small, Medium, Large, Extra-Large on order form)
White design on a dark blue shirt.

POSTER (*Oct., 1980 BYTE* cover) 602 - $5 1# Last 10

FORTH-83 HANDY REFERENCE CARD 683 - free

FORTH-83 STANDARD 305 - $15 1#
Authoritative description of Forth-83 Standard. For reference, not instruction. *83 pgs*

BIBLIOGRAPHY OF FORTH REFERENCES 340 - $18 2#
(3rd ed., January 1987)
Over 1900 references to Forth articles throughout computer literature. *104pgs*

JFAR BACK ISSUES

Volume 2,#2,#3,#4 JFAR (1984) 722 - $45 2#
#2 - Laser beveling, introductory robotics, Forth-based robotic vehicle, Kitt Peak Forth. #3 - Forth meets Smalltalk, Stack usage, number crunching. #4 - Forth in hardware, VAX & 79 Standard, extended address space, local words in Forth, binary search.

Volume 3, #1-4 JFAR (1985) 723 - $65 3#
#1 - Real-time control, stack frames. #2 - 1985 Rochester Conference Proceedings, MAGIC/L #3 - Microcode assembler, heap data structure, object-oriented extentions to Forth, discrete event simulation. #4 - numerical control, exception handling, state sequence handlers.

Volume 4, #1-4 JFAR (1986-1987) 724 - $65 3#
#1 -Expert systems in Forth: natural language parsing, Microcomputer-based medical diagnosis system, FORTES polysomnographer, FORPS. #2 - 1986 Rochester Conference Proceedings. #3 - REPTL, stand alone Forth system, compiling Forth, Julian Day numbers, abstracts '86 FORML conference. #4 - Embedding of languages in Forth, Forth-based Prolog for real-time expert systems.

Volume 5, #1-4 JFAR (1988-1989) 725 - $65 3#
#1 - 1987 Rochester Conference Proceedings. #2 - Mathematics, ANS standard, exception handling, logarithmic number representation, 32-bit RTX chip prototype. #3 - From Russia with Forth, knowledge engineering, symbolic stack addressing. #4 - Forth processors, parallel Forth, arithmetic-stack processor, architecture of the SC32 Forth engine, error-free statistics in Forth.

Rochester, 1981 751 - $25 2#
Standards Conference

Rochester, 1990 760 - $30 2#
Embedded Systems

MORE ON FORTH ENGINES

Volume 10 January 1989 810 - $15 1#
RTX reprints from 1988 Rochester Forth conference, object-oriented cmForth, lesser Forth engines. *87 pgs*

Volume 11 July 1989 811 - $15 1#
RTX supplement to *Footsteps in an Empty Valley*, SC32, 32-bit Forth engine, RTX interrupts utility. *93 pgs*

Volume 12 April 1990 812 - $15 1#
ShBoom Chip architecture and instructions, neural computing module NCM3232, pigForth, binary radix sort on 80286, 68010, and RTX2000. *87 pgs*

Volume 13 October 1990 813 - $15 1#
PALs of the RTX2000 Mini-BEE, EBForth, AZForth, RTX-2101, 8086 eForth, 8051 eForth. *107 pgs*

Volume 14 814 - $15 1#
RTX Pocket-Scope, eForth for muP20, ShBoom, eForth for CP/M & Z80, XMODEM for eForth. *116 pgs*

Volume 15 815 - $15 1#
Moore: new CAD system for chip design, a portrait of the P20; Rible: QS1 Forth processor, QS2, RISCing it all; P20 eForth software simulator/debugger. *94 pgs*

Volume 16 816 - $15 1#
OK-CAD System, MuP20, eForth system words, 386 eForth, 80386 protected mode operation, FRP 1600 - 16-Bit real time processor. *104 pgs*

DR. DOBB'S JOURNAL

Annual Forth issue, includes code for various Forth applications.
Sept. 1982 422 - $5 1#
Sept. 1983 423 - $5 1#
Sept. 1984 424 - $5 1#

FORTH INTEREST GROUP

P.O. BOX 2154 OAKLAND, CALIFORNIA 94621 510-89-FORTH 510-535-1295 (FAX)

Name _____ Phone _____
Company _____ Fax _____
Street _____ eMail _____
City _____
State/Prov. _____ Zip _____
Country _____

U.S. Domestic Postage Rates	Surface	2 day Priority
	$1.00/#	$1.50/#
International Postage Rates	Surface	AIR MAIL
	All /#	1-4 #s/#	>4 #s/#
Canada, Mexico	$1.00	$2.00	$1.30
Other Western Hemisphere	$1.00	$3.25	$2.25
Europe	$1.00	$6.00	$4.50
Other International	$1.00	$8.00	$6.00

Item #	Title	Qty.	Unit Price	Total	#

☐ CHECK ENCLOSED (Payable to: FIG)
☐ VISA/MasterCard Expiration Date _____
Card Number _____
Signature _____

Sub-Total		
10% Member Discount, Member # _____	( )	#s times rate
**Sales Tax on Sub-Total (CA only)		
Postage: Rate _____ x #s		
*Membership in the Forth Interest Group ☐New ☐Renewal $40/46/52		
MEMBERSHIP Total		

*MEMBERSHIP IN THE FORTH INTEREST GROUP

The Forth Interest Group (FIG) is a world-wide, non-profit, member-supported organization with over 1,500 members and 40 chapters. Your membership includes a subscription to the bi-monthly magazine *Forth Dimensions*. FIG also offers its members an on-line data base, a large selection of Forth literature and other services. Cost is $40 per year for U.S.A. & Canada surface; $46 Canada air mail; all other countries $52 per year. This fee includes $36/42/48 for *Forth Dimensions*. No sales tax, handling fee, or discount on membership.
When you join, your first issue will arrive in four to six weeks; subsequent issues will be mailed to you every other month as they are published—six issues in all. Your membership entitles you to a 10% discount on publications and functions of FIG. Dues are not deductible as a charitable contribution for U.S. federal income tax purposes, but may be deductible as a business expense.

MAIL ORDERS:
Forth Interest Group
P.O. Box 2154
Oakland, CA 94621
PHONE ORDERS:
510-89-FORTH Credit card orders, customer service. Hours: Mon–Fri, 9–5 p.m.

PAYMENT MUST ACCOMPANY ALL ORDERS

PRICES: All orders must be prepaid. Prices are subject to change without notice. Credit card orders will be sent and billed at current prices. Checks must be in U.S. dollars, drawn on a U.S. bank. A $10 charge will be added for returned checks.

POSTAGE: All orders calculate postage as number of #s times selected postage rate. Special handling available on request.

SHIPPING TIME: Books in stock are shipped within seven days of receipt of the order.

**** CALIFORNIA SALES TAX BY COUNTY:**
7.5%: Sonoma; **7.75%:** Fresno, Imperial, Inyo, Madera, Orange, Riverside, Sacramento, San Benito, Santa Barbara, San Bernardino, San Diego, and San Joaquin; **8.25%:** Alameda, Contra Costa, Los Angeles San Mateo, Santa Clara, and Santa Cruz; **8.5%:** San Francisco; **7.25%:** other counties.

For faster service, fax your orders: 510-535-1295

```
            70 5 at ." omemqe d"
            70 6 at ." pssaua r"
            70 7 at ." rsbvev g"
            70 8 at getsb
            8 base @ 2 base ! -rot .r base !
\ after restoring cursor, wait for a short while,
\ so we have time to read the status byte
            restcursor 1 ms
        then
    again ;


comment: /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
        This routine starts the data collection (COLLECT) and screen display
(TASK1) tasks running in the background.  Enter a file name in which to save
data after the word XTAL. If the file name already exists, XTAL will
ask to overwrite.  Type 'fin' when done; this will stop background
tasks, and close data file.
        /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
comment;

: getfname ( | filename -- )
        xtalhandle !hcb
        xtalhandle hopen 0= if
                ." file already exists.  Overwrite (Y/n)?"
                key upc 'N' = if abort" No. Alright." cr then
        then
        xtalhandle hcreate 0 <> abort" error on hcreate"
        ;

: xtal ( | filename -- )
        getfname
        datacnt off gooddata off
        get-time starttime 2!
        task1 wake collect wake
        cr ." Type FIN when you want to stop collecting..." cr
        ." or SINGLE if you want to pause...then MULTI" cr
;

: fin \ finished with collecting
        task1 sleep collect sleep
        xtalhandle hclose drop local
        ;

\ .used
: checkfreq ( -- )
    gpib0handle ibfind =: gpib0
    pm6680handle ibfind =: pm6680
    begin
        cr ." Checking PM6680 frequency counter..." pm6680 awake? not while
        ." Please turn it on, and press a key..." key drop
    repeat ." it's responding ok"
    cr initfreq
    ;
\ gpib0handle ibfind =: gpib0
\ pm6680handle ibfind =: pm6680
checkfreq
\ initfreq
multi
\ task2 wake task3 wake \ start error reporting, and status byte
```

Forth in Search of a Job

Donald Kenney
Canton, Michigan

I've read a fair amount in *Forth Dimensions* about the lack of Forth usage in larger engineering and programming shops. As a reformed software manager, I'd like to explain some of the reasons.

I'm not claiming that software or engineering management is a rational environment. In point of fact, it is a very wacky environment, and far too many of the participants are none too tightly wrapped. Nonetheless, they are generally very bright, and even the nut cases fit the punch line of an old joke: they are crazy, not stupid. There are reasons they don't use Forth.

1. Programming is not a one-time operation. Large systems require integration. Almost any software, once released, is going to require maintenance—enhancements and bug fixes. It is impossible to predict which projects will need how many people more than a few weeks in advance. If a shop tries to work in 22 dialects of 13 programming languages, the chance of having a suitably trained person on hand when a need arises isn't high. Most shops try to work in as few programming languages as possible.

2. The choice of programming language doesn't look anywhere near as important to a manager as it does to a programmer. The manager knows that most of the budget goes to requirements analysis, design, testing, configuration control, overhead, facility, product maintenance, manual production, and endless meetings. Programming *per se* is a relatively minor cost, and adding a little expense there doesn't hurt a lot. I've forgotten the conventional wisdom on this—the figure 10% of total costs comes to mind, but that's low, because it excludes design and test activities of programmers. Anyone who cares should check Barry Boehm's *Software Engineering Economics.* Anyway, a non-optimal language choice doesn't bother the manager much—especially if the (very real) costs of bringing a new language into the operation can be avoided.

3. Forth is a fringe language. If one needs to find additional experienced programmers in a hurry, who is the manager going to call? And what will they cost? For C, Cobol, Fortran, or BASIC, the friendly neighborhood body shop can probably find some reasonable local folks. The manager may even have the luxury of choosing between several qualified candidates. For Forth, the nearest available resource is likely to be in Fargo, North Dakota. There is a real danger of paying a premium price for services and a fortune in travel expenses.

4. Getting beat up by the suits in the front office is no fun. Why propose using a language they've never heard of? At the very least, a good story is needed. While most managers could probably concoct a suitable fairy tale, why would they want to?

5. Much as I like Forth, I'm by no means convinced that Forth is a good choice for a big system language. Yes, the code will probably be great and the pieces will be beautiful to behold. But the system design (if any) will probably have been bungled. It usually is (generally because of management impatience, not designer ineptitude). Those pretty pieces are probably going to have to be trimmed and reworked to build a working system. That means people working with each other's code. Forth is often hard to read, and I'd be concerned about ending up with people blowing each other out of the water with ill-considered, low-level changes to each other's code. I'm not sure how to handle shared data in a large Forth system. Not that I can't devise a way. But how do I know it will work?

If I'm not sure how well this good one/two person language is going to scale to eight- or 20-person projects, how do you think the average manager feels? Why should they experiment when they know C or Fortran will do the job?

6. If the programming shop has reasonable controls, configuration management, library control, etc., the managers should be concerned about adapting them to Forth. In point of fact, the controls probably won't adapt. The shop will have to develop new procedures from scratch. That's a lot of work.

So, it's hopeless? There's no room for Forth in big operations?

Not so. But the only place Forth is likely to penetrate is in niches where nothing already in place will suffice, or where the product occupying the niche is truly unsatisfactory. Forth is marketable as a language to program microcontrollers, or for environments with restricted address space. It is marketable as a general replacement for assembly language in environments that don't require really high performance but which don't have the resources to support a conventional, higher-level language. It won't displace tiny Cs and BASICs where the shop also uses C or BASIC for larger projects. Maybe Forth can be sold as a prototyping language. But Forth is not likely to replace mainline programming languages. Most people don't see any need for another mainline programming language. If Ada, with the full support of the Department of Defense, can't penetrate the DOD's captive contractor base, what chance does Forth stand in penetrating a decidedly non-captive industry base?

Integer Date Calculations

Richard de Rozario
Randwick, NSW, Australia

These days, routines for calculating dates fall into the category of "classics." When I needed some words for this in the Pygmy version of Forth, I first looked through back issues of *Forth Dimensions*. (I mean, why re-invent the wheel, right?) Allen Anway (*FD* IX/1) and Matt Wilson (*FD* IX/3) provided a good starting point. However, I was looking for a calculation that would result in a two-byte day number, using integer operations. Neither demand was set in concrete, but they prompted me to pick to pieces the actual formulas used in calculating dates (most articles I have seen draw on un-elaborated formulas from other sources). In doing so, I discovered some modifications that are useful in strictly integer Forths, like Pygmy.

The main idea is to convert a date into a day number, and vice versa. That allows you to calculate things like the number of days between two dates. The way to figure a day number for a particular date is to add all the days in years passed, add all the days in months passed, and add the number of days passed in this month. So, if your starting point is 1 Jan. 1900 (day 1) and you wish to figure the day number for 1 June 1993, you would add all the days till 1993 (33968), all the days till June (151), plus 1, resulting in day number 34120.

There are two hard parts to this. One is that only some years have a leap day; the other problem is that the number of days in each month varies. The two problems are compounded by the fact that the leap day is in February—right in the middle of things.

A trick used in all formulas is to attach January and February to the end of last year, and make this year start in March. This will put the possible leap day at the end of the year. In other words, the leap day will be added when you calculate the days of years passed (which is a lot simpler than keeping track of it in the middle of your month calculations). So March becomes month number 0, and January and February become months 10 and 11, respectively (and subtract 1 from the year if you're dealing with Jan. or Feb.). A few examples:

1 Jan 92	=	1 10 91
29 Feb 92	=	29 11 91
1 Mar 92	=	1 0 92
1 Apr 92	=	1 1 92

Note that this means your base year (e.g., 1900) also doesn't start in January, but in March. So day number 1 would be 1 March 1900.

With this out of the way, we can proceed with the actual calculation of the day number. First, adjust the date as described above, then calculate the number of days in years passed as follows (remember, this will automatically include any leap days): N = Int(365.25 * y)

or in Forth:

```
( y -- n)
DUP 365 *
SWAP 25 * 100 / +
```

In our time frame of 1900 to 2099, we don't have to bother calculating exceptions to leap years (i.e., divisible by 100 is not a leap year, but divisible by 400 is). All years divisible by four are leap years in this period. On the other hand, if you wish to extend the calculations, it won't be hard to do.

We are now left to calculate the days in months passed. In our re-ordered months, the number of days accumulate like this:

#	Month	Days	Sum
0	Mar	31	0
1	Apr	30	31
2	May	31	61
3	Jun	30	92
4	Jul	31	122
5	Aug	31	153
6	Sep	30	184
7	Oct	31	214
8	Nov	30	245
9	Dec	31	275
10	Jan	31	306
11	Feb	28	337

So the goal is to find a number that, multiplied with our month number, will tell us the sum total of days that have come before. Anway uses 30.59, which works when rounded on the first decimal. For example, the number of days before August (month 5) would be: 5 * 30.59 = 152.95 –> 153.

When using integer calculations only, I find 30.4 (or 304) a better number. The result of multiplication by 30.4 is off by a small amount, but the differences form a pattern that is easily compensated for. The outcomes look like this:

#	Month	Days	Sum	M*30.4	diff
0	Mar	31	0	0	0
1	Apr	30	31	30	1
2	May	31	61	60	1
3	Jun	30	92	91	1
4	Jul	31	122	121	1
5	Aug	31	153	152	1
6	Sep	30	184	182	2
7	Oct	31	214	212	2
8	Nov	30	245	243	2
9	Dec	31	275	273	2
10	Jan	31	306	304	2
11	Feb	28	337	334	3

The compensated integer calculation for days of months passed would be as follows (where m is the month number and \ means integer divide):

if m=0, N = 0
else N = m * 304 \ 10 + m\6 + m\11 + 1

After we have calculated the days of years passed and the days of months passed, all we have to do is add the day of the date. The result is the date converted into a day number.

The reverse calculations are similar. We now have a day number and want to calculate the date components. First, divide by 365.25 to get the year. Next, calculate the days of years passed and subtract that from the day number. Next divide by a "compensated fraction" to get the month. This is a different compensated fraction than the one used to *get* the day number. The main reason is that we now work from the day number, not from the number of days in months passed. For example, the 1st of June is day number 93, but the days of (re-ordered) months passed is 92.

So we divide by 30.41 instead of 30.4. Also, before we divide we should take away those compensations that we added when we calculated the day number. Again, because we now work from the day number (and not from the date), we use different divisors to accomplish this. The complete integer number-to-month calculation is as follows:

m = (N - N\184 - N\337 - 1) * 100 \ 3041

```
\ date calculation. ref: Forth Dimensions XV/1.
( from date to daynum. NOTE: only good for 1900-2079)
: PREPY ( y-z)      DUP 1900 > IF 1900 - THEN ;
: PREPM ( m-n)      DUP 3 < IF 12 + THEN 3 - ;
: PREPD ( dmy-dnz) PREPY SWAP PREPM SWAP OVER 9 > IF 1- THEN ;
: YDAYS ( y-z)      DUP 365 * SWAP 25 * 100 / + ;
: MDAYS ( m-n)      DUP IF DUP 304 * 10 / SWAP DUP 11 /
                    SWAP 6 / + + 1+ THEN  ;
: DAYNUM  ( dmy-g) PREPD YDAYS SWAP MDAYS + + ;
: WEEKDAY ( g-n)    5 - 7 UMOD ; ( 0=Mon, 1=Tue, etc.)


CODE UM* ( u u - d)
  AX POP,  BX MUL,  AX PUSH,  DX BX MOV,  NXT,   END-CODE
CODE UM/ ( l h u - u )
  DX POP,  AX POP,  BX DIV,  AX BX MOV,   NXT,   END-CODE

: N2Y       ( g-n) 4 UM* 1461 UM/ ( ie. / 365.25) ;
: D2M       ( n-m) DUP 184 / OVER 337 / + 1+ - 100 3041 */ ;
: YADJ    ( yn-ym) DUP 0= IF DROP 1- 366 THEN ;
: MADJ      ( n-m) 3 + DUP 12 > IF 12 - THEN ;
: NUMDAY ( g-dmy) DUP N2Y DUP YDAYS ROT SWAP - YADJ
                  DUP D2M DUP MADJ SWAP MDAYS ROT SWAP -
                  SWAP ROT OVER 3 < IF 1+ THEN 1900 + ;


VAR: MLEFT VAR: MTOP VAR: #CELLS VAR: #ROWS
5 TO MLEFT 3 TO MTOP 7 TO #CELLS 5 TO #ROWS

: TOP       ( -) CUR@ NIP MTOP SWAP AT ;
: DOWN      ( -) CUR@ SWAP 1+ SWAP AT ;
: HOME      ( -) CUR@ DROP MLEFT AT ;
: .ROW   ( g-h) #CELLS FOR DUP NUMDAY 2DROP 3 .R SPACE 1+ NEXT ;
: .ROWS ( gn-h) FOR .ROW DOWN HOME NEXT ;
: .PAGE    ( g-) 0 OVER WEEKDAY - + TOP HOME #ROWS .ROWS ;
: .HEADER  ( -) MTOP 1- MLEFT AT
                ." Mon Tue Wed Thu Fri Sat Sun" ;
: .CAL ( dmy-h) .HEADER DAYNUM .PAGE ;

( usage: 25 12 92 DAYNUM WEEKDAY . )
(        25 12 92 DAYNUM NUMDAY . . . )
( to print a small calender type d m y .CAL)

\  For systems without them, here are
\  implementations of VAR: and TO
: VAR: ( -)  CREATE ] DOES> ( a-) @ ;
: TO   ( n-) ' >BODY ! ;
  COMPILER : TO  ( n-)  $15A , ' >BODY , ['] ! , ; FORTH
```

By the way, when you subtract the days of years passed, you may get a remainder of exactly zero. That means this is the day number of a leap day. Make a final compensation of subtracting 1 from the year and altering the remaining days to 366. Then the rest of your calculation will automatically come out to February 29.

The source code for doing these calculations in Pygmy Forth is given here, including some words to print a calender. Note that Pygmy Forth uses UMOD instead of the 83-Standard UM/MOD. I have also added assembly code for UM* and UM/ which 83-Standard Forths won't need. Use the built-in UM* and UM/MOD instead. Definitions for VAR: and TO are included for those who don't have them.

 The author is a PC software trainer living in Sydney, Australia.

Book Review

Forth: The New Model

by Jack Woehr

$45, M&T Publishing
Available from the Forth Interest Group and other book sellers.

Reviewed by Charles Curley
Gillette, Wyoming

If you didn't gag at the price of *Forth: The New Model* and skip to the next article, congratulations. The book may well be worth it to you. For one thing, it includes an MS-DOS 360K 5.25" disk with source to the exercises in the book, and a copy of Martin Tracy's ZENForth. If you are one of the people who must wrestle with the dp-ANS Forth in any way, this book will save you much frustration and grief.

I have tried, on occasion, to read the various dp-ANS Forth versions, and have found them to be obtuse, confusing, and sometimes self-contradictory. This kind of language would get an IRS manual writer reprimanded for writing too obscurely.

To truly get a grasp on dp-ANS Forth, it helps to be a giant squid. In one tentacle, you hold a copy of the draft proposed standard. Until it is approved and cast in solid hydrogen, the latest version will have to do. Alternatively, get a copy of the version to which your standard System was written. In another tentacle, you need a copy of this book. In the third tentacle, you need a standard Forth system's documentation. In tentacles four and five, a pen and a new pad of Post-It[1] notes, so you can cross-reference all the documentation! That should leave several tentacles free for pounding on the keyboard, drinking cola, grabbing submarines as they go by, etc. Now, as Captain Nemo said to Ned Land, let's get kraken.

If you are planning to use dp-ANS Forth, get this book. If you haven't decided yet, this book will help to make your decision.

The first thing to do is grab a Post-It note, label it "Data Types," and use it to flag table 3-1 on page 48 of this book. You will need that to decipher the hieroglyphics in the standard's stack diagrams (yes, the standard has rules for them, too).

Woehr's book is a major improvement on the standard. Where the standard has all the crisp, pellucid precision of Houston Shipping Channel water, Woehr's book is up to the standards of a mineral hot springs bath. At least you can dive in with reasonable assurance that you will be able to climb out with your sanity more or less intact.[2] And it might even be good for you.

Forth: The New Model tackles a specific standard system, Vesta's Vesta Forth Standard Edition.[3] It tackles it systematically, covering logical groupings of words together. It discusses control structures together; logical and bitwise operators together; etc. To get the most out of this book, read up on a subject in the book, refer to the glossary, and then tackle the standard and your system's documentation for details and points the book may have missed.

To give an example of how the three documents interact, let's look at `DO ... LEAVE ... LOOP`. We know that the following works, because Woehr tells us so:

```
: LEMME-OUT
  100000 0 DO
  I . LEAVE LOOP ;
```

This is completely useless code. Closer to the real world, will the following compile?

```
: LEMME-OUT
  100000 0 DO
  I 1000 = IF LEAVE THEN
  I . LOOP ;
```

If it will compile, will the phrase `I .` be executed after the word *LEAVE* is executed? Woehr's text is silent on both questions. If you can't find the answer in the standard (good luck), you may have to hope that the way the documentation says your system works is the way every other dp-ANS Forth works. Good luck.[4]

Can the answer to the first question be deduced? The standard—and Woehr—makes much of a theoretical stack for control structure compilation, called the control flow stack. Without using (gag) control flow stack manipulators (`CS-ROLL`, `CS-PICK`, etc.), crossing the nesting imposed by this stack is illegal. If *LEAVE* is part of the *DO ... LOOP* structure, the second version of the sample code is illegal.

But the stack diagrams for the looping words indicate that they don't use the control flow stack, they use the return stack. So we're using two different stacks and we're okay, right? Wrong: there is nothing that prevents the implementor from using the return stack for the control flow stack!

The answer to the second question is stated in Woehr's

glossary, but not in the body of the book.

In the case of most Forth words, the casual user may refer to Woehr's book and let the standard go. Usually, words are covered in better detail than the example I have given above. To be more precise, the kind of detail I would have liked to see with *DO ... LEAVE ... LOOP* isn't necessary with most words, even in dp-ANS Forth.

In almost all cases, Woehr shows words being used, so the reader can see how they are used. This sort of teaching by example is far beyond the scope of the standard, but essential for its understanding.

For example, some folks may find the concepts behind DEFER ... IS obscure. Where the standard just lays them out, take 'em or leave 'em, Woehr gives working source code defining them and using them in a small application. And the source code is in a text file on the disk.

Unfortunately, Woehr has fallen into a trap common to writers of books on Forth. The simpler the concept, the more a write-up explains it. The more complex the concept, or the newer to the reader, the more write-up is necessary. I found Woehr's write-up of simple concepts like *DUP* over-long and verbose (but, knowing that neoforthwrights will be reading this book, I do not consider that a negative). I found the discussion of (to me) medium-level concepts like CREATE ... DOES> or DEFER ... IS to be just about right for me. Neoforthwrights may wish for more detail.

Unfortunately, his discussion of THROW ... CATCH (or is it CATCH ... THROW?), complete with sample code showing how to use it, left me almost completely in the dark. I say "almost" only because, prior to reading the book, I had no idea that these two words existed. I now know that they do exist, and that they have something to do with error handling. Thanks, Jack.

This is a problem to which almost all technical writing is prone. If the concepts are easy to understand, they can be written up quickly, and the writer can move on to more advanced subjects. But the writer (we hope) understands the advanced subjects, whereas the reader may not. So there is a temptation for the writer to write an explanation which would be suitable to explain the concept—to the writer.

Like Gandalf says in Tolkien's *The Lord of the Rings*, older folks tend to talk to the wisest person in the room: themselves.

In addition to explaining how words work, both in isolation and in the context of sample programs, Woehr also discusses the philosophy behind some of the code and that behind parts of the standard. In support of his points, Woehr refers to such great twentieth century philosophers as Groucho Marx and Douglas Adams. He also cites knowledgeable Forthwrights, such as Charles Moore, Dudley Ackerman, and Charlie Johnsen.

Even if Woehr says that something is so, is it in fact so? No. Where Woehr and the standard conflict, the standard, of course, overrules. This means that both system implementors and standard program authors will have to use the standard, in all its opacity, as the final authority. We find the same thing over and over again in law: too many lawyers read only the case notes in the statute books, and don't read the actual cases. The case notes aren't the law, the cases are. Woehr's book isn't the standard, the standard is.

Unfortunately, the same laziness that leaves some lawyers relying on the case notes will leave some implementors and programmers relying on Woehr's book, and they (and their customers) will be bitten by any differences that may exist.

Some of Woehr's prejudices show through. For example, his near tirade against using screens is silly. His suggestion that one go out and buy a "good, mouse-driven text editor that runs in a window" instead of using a block editor is the kind of fatuous "We're from Microsoft and we know more about your application that you do" tripe of which programming in Forth should have cured him long ago. These are two points on which he should have taken Elizabeth Rather's oft-stated advice: put a cork in it, Jax.[5]

In praise of Jack Woehr, let me say that he has a sense of humor and isn't afraid to let it out to run loose. Good technical writing need not—indeed, should not—be boring. A good sense of humor, well deployed, makes a technical book much more readable, as Leo Brodie showed. Readers should not confuse humor with ignorance or lack of professionalism. Indeed, often a well-placed laugh will provide the reader with the mnemonic necessary to remember a point. Thank you, Jack!

Absolutely essential in this book is the excellent glossary of the Vesta Standard Forth. It includes details about each word, and gives stack diagrams as needed. All Forths should include such a document, and vendors should look at this glossary as a point of comparison for their own documentation.

There is an index, but it is sparse. The index has some quirks, like an entry for HP calculators, pages 3–4, and another entry for Hewlett-Packard calculator, page 74. Word names only index the discussion of that word. I would also like to see the index refer to other places where words are used, so the reader can look at examples.[6]

The bottom line on this book is this: If you are planning to use, or considering using, dp-ANS Forth, *get this book.* If you haven't decided yet whether to go to dp-ANS Forth, this book will help you to make your decision. Ideally, get your boss to shell out the $45 plus state theft, shipping, and handling. If you are your boss, you should almost certainly shell out for it anyway. The book clarifies and annotates the standard, and will save hours of frustration, thereby paying for itself.

For the rest of us, it's too expensive. I would not have bought it solely for my own library.

---

[1] Trademark of 3M, MR. I use the 38 x 50 mm size, #653.

[2] This *is*, after all, a book on Forth.

[3] Know, then, that Jack Woehr works for Vesta, and is the author of Vesta Standard Forth. *Now* do you understand recursion?

[4] You could also experiment with your system. But one gets the idea that would be considered cheating.

[5] Quoted in Woehr, page xvii.

[6] And, somebody, please tell the publisher of the next edition to make the end notes into footnotes. Jack's notes are worth reading, and footnotes are easier.

Charles Curley is a paleoforthwright living in Gillette, Wyoming. He is the founder of the Forth Non-Standards Team.

HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

By now you know that HS/FORTH gives you more speed, power, flexibility and functionality than any other language or implementation. After all, the majority of the past several years of articles in Forth Dimensions has been on features found in HS/FORTH, often by known customers. And the major applications discussed had to be converted to HS/FORTH after their original dialects ran out of steam. Even the public domain versions are adopting HS/FORTH like architectures. Isn't it time you tapped into the source as well? Why wait for second hand versions when the original inspiration is more complete and available sooner.

Well, it was a dirty job, but we finally had to do it. Now you can run lots of copies of HS/FORTH from **Microsoft Windows** in text and/or graphics windows with various icons and pif files available for each. Talk about THE tool for hacking Windows! But, face it, what I really like is cranking up the font size so I can still see the characters no matter how late it is. Now that's useful.

Good news, we've redone our **DOCUMENTATION!** The big new fonts look really nice and the reorganization makes all that functionality so much easier to find. Thanks to excellent documentation, all this awesome power is now relatively easy to learn and to use.

Naturally we continue tweaking and improving the internals, but by now the system is so well tuned that these changes are not individually of any significance. They just continue to improve performance a bit at a time, and enhance error detection and recovery. **Update to Revision 5.0**, including new documentation, from all 4.xx revisions is $99. and from really old systems the update is $149.

And since Spring is coming, IT IS TIME FOR OUR **SPRING SALE**. Thru the end of May you get to pick two extra utility packages free for each Professional or Production Level system purchased, or get a free Online Glossary with help file utility with each Personal Level system purchased.

HS/FORTH runs under MSDOS or PCDOS, or from ROM. Each level includes **all** features of lower ones. Level upgrades: $25. plus price difference between levels. Source code is in ordinary ASCII text files.

HS/FORTH supports megabyte and larger programs & data, and runs as fast as 64k limited Forths, even without automatic optimization -- which accelerates to near assembler language speed. Optimizer, assembler, and tools can load transiently. Resize segments, redefine words, eliminate headers without recompiling. Compile 79 and 83 Standard plus F83 programs.

PERSONAL LEVEL $299.
NEW! Fast direct to video memory text & scaled/clipped/windowed graphics in bit blit windows, mono, cga, ega, vga, all ellipsoids, splines, bezier curves, arcs, turtles; lightning fast pattern drawing even with irregular boundaries; powerful parsing, formatting, file and device I/O; DOS shells; interrupt handlers; call high level Forth from interrupts; single step trace, decompiler; music; compile 40,000 lines per minute, stacks; file search paths; format to strings. software floating point, trig, transcendental, 18 digit integer & scaled integer math; vars: A B * IS C compiles to 4 words, 1..4 dimension var arrays; **automatic optimizer delivers machine code speed.**

PROFESSIONAL LEVEL $399.
hardware floating point - data structures for all data types from simple thru complex 4D var arrays - operations complete thru complex hyperbolics; turnkey, seal; interactive dynamic linker for foreign subroutine libraries; round robin & interrupt driven multitaskers; dynamic string manager; file blocks, sector mapped blocks; x86&7 assemblers.

PRODUCTION LEVEL $499.
Metacompiler: DOS/ROM/direct/indirect; threaded systems start at 200 bytes, Forth cores from 2 kbytes; C data structures & struct+ compiler; MetaGraphics TurboWindow-C library, 200 graphic/window functions, PostScript style line attributes & fonts, viewports.

ONLINE GLOSSARY $ 45.

PROFESSIONAL and PRODUCTION LEVEL EXTENSIONS:

FOOPS+ with multiple inheritance **$ 79.**
TOOLS & TOYS DISK $ 79.
286FORTH or **386FORTH $299.**
16 Megabyte physical address space or gigabyte virtual for programs and data; DOS & BIOS fully and freely available; 32 bit address/operand range with 386.
ROMULUS HS/FORTH from ROM **$ 99.**

Shipping/system: US: $9. Canada: $21. foreign: $49. We accept MC, VISA, & AmEx

Two Examples of Numbers: Calendars & The Game of Life

C.H. Ting
San Mateo, California

In this lesson, we will use two rather involved examples to illustrate the process of composing Forth instructions to solve real problems, using the set of Forth instructions we've learned in the last three lessons.

Calendars

We want to print weekly calendars for any month in any year. (Actually from 1950 to, maybe, 2050.) The days are printed in columns (Sunday, Monday, and so forth). This problem is not as simple as it first looks, because many things have to be taken into accout: the leap year, the number of days in a month, and the day of the week for the first day in a month.

We approach this problem by computing first the Julian date of January 1 of the year (YYYY) in which we are interested. Then we compute the number of days from January 1 to the first day of the month (MM) of interest. Adding the Julian date of the year, the number of days until the first day of the month, and the day (DD) in the month, we get the Julian date of the day specified by (DD MM YYYY). The day of the week is, then, the remainder of the Julian date divided by seven.

Let's use January 1, 1950 as day zero of our modified Jualian calendar. It also happens to be a Sunday. There are 1461 days in four years, with 366 days in a leap year. Thus, the Julian date of any year can be computed easily by YEAR and stored in the variable JULIAN. If this year is a leap year, the variable LEAP contains a one; otherwise, it contains a zero.

It is more difficult to compute how may days there are, in a given year, until the first day of any month. It is done in the instruction FIRST. FIRST pops the number of the month (one for January, two for February, etc.) and returns the number of days there are in a year from January 1 to the first

```
VARIABLE JULIAN ( Julian date of 1st of a year, from Jan 1, 1950)
VARIABLE LEAP   ( 1 for a leap year, 0 otherwise. )
1461 CONSTANT 4YEARS ( number of days in 4 years )

: YEAR ( YEAR --, compute Julian date and leap year )
       1949 - 4YEARS 4 */MOD          ( days since 1/1/1949 )
       365 - JULIAN !                 ( 0 for 1/1/1950 )
       3 =                            ( modulus 3 for a leap year )
       IF     1 LEAP !                ( leap year )
       ELSE   0 LEAP !                ( normal year )
       THEN ;

: FIRST ( MONTH -- 1ST, 1st of a month from Jan. 1 )
        DUP 1 =
        IF DROP 0                          ( 0 for Jan. 1 )
        ELSE    DUP 2 =
            IF      DROP 31            ( 31 for Feb. 1 )
            ELSE    DUP 3 =
                    IF      59 LEAP @ +     ( 59/60 for Mar. 1 )
                    ELSE    4 - 30624 1000 */
                            90 + LEAP @ +   ( Apr. 1 to Dec. 1 )
                    THEN
```

```
                THEN
        THEN
        ;

: DAY ( DD MM YYYY -- JULIAN-DAY )
        YEAR                        ( Compute JULIAN and LEAP)
        FIRST + 1-                  ( add DD to 1st of the month )
        JULIAN @ +                  ( add to Jan. 1 of the year )
        ;

: STARS 0 DO 42 EMIT LOOP ;                ( form the border )

: header ( n -- )                          ( print title bar )
        cr cr 26 stars space
        case      1 of ."  January " endof
                  2 of ." February " endof
                  3 of ."   March  " endof
                  4 of ."   April  " endof
                  5 of ."    May   " endof
                  6 of ."   June   " endof
                  7 of ."   July   " endof
                  8 of ."  August  " endof
                  9 of ." September" endof
                 10 of ."  October " endof
                 11 of ." November " endof
                 12 of ." December " endof
                 DROP
        endcase
        space 27 stars cr cr
        ."      SUN     MON     TUE     WED     THU     FRI     SAT"
        cr cr                              ( print weekdays )
        ;

: BLANKS ( MONTH -- )                ( skip days not in this month )
        FIRST JULIAN @ +             ( Julian date of 1st of month )
        7 MOD 8 * SPACES ;           ( skip colums if not Sunday   )

: .DAYS ( MONTH -- )                ( print days in a month )
      DUP FIRST                     ( days of 1st this month )
      SWAP 1 + FIRST                ( days of 1st next month )
      OVER - 0                      ( loop to print the days )
      DO I OVER +
        JULIAN @ + 7 MOD            ( which day in the week? )
        0= IF CR THEN               ( start a new line if Sunday )
        I 1 + 8 U.R                 ( print day in 8-column field )
      LOOP
      DROP ;                        ( discard 1st day in this month )

: MONTH ( N -- )                    ( print a month calendar )
      DUP
      HEADER DUP BLANKS             ( print header )
      .DAYS ;                       ( print days   )

: CALENDAR ( YEAR --- )             ( print year calendar )
      YEAR                          ( compute JULIAN and LEAP )
      13 1 DO I MONTH LOOP          ( print 12 month calendars )
      CR CR 64 STARS ;              ( print last border )
```

day of the month. February 1 is the 31st day of the year. March 1 is either the 59th or 60th day, depending on whether it is a leap year; April 1 is the 90th or 91st day of the year, and so forth. A neat formula is used in FIRST to compute the 1st of May and all the months it.

The instruction DAY takes a day DD, a month MM, and a year YYYY from the stack, and returns the Julian date of that day to the stack. The Julian date can be used to determine which day it is in a week by taking its modulus of 7. Sunday has a modulus of 0, and Saturday, 6.

.DAY prints the days in a month as specified by the month number on the stack, with the columns aligned to the days in a week. This formatted display is enhanced, by the instruction MONTH, with borders, name of the month, and names of the days. For example, type

```
1992 YEAR 7 MONTH
```

to display the July calendar of 1992. The instruction CALENDAR displays the 12-month calendar of any year, specified by the year number on the stack. You will not see much on the computer screen, but you can use it if you print it on your line printer.

```
1992 YEAR CALENDAR
```

To print the calendar, type

```
PRINT
1992 YEAR CALENDAR
```

PRINT is a special F-PC instruction to send screen output to printer.

Example One
The True Julian Date

```
: JULIAN-DATE ( DD MM YYYY -- d, Julian date as a double integer)
        >R                      ( save YYYY on return stack)
        DUP 9 + 12 /            ( 0 for Jan/Feb, 1 for others)
        R@ + 7 * 4 / NEGATE     ( take 1.75 days out for each year)
                                ( 365.25=367-1.75 )
        OVER 9 + 12 / NEGATE
        R@ +
        100 / 1 + 3 * 4 / -     ( leap days generated by centuries)
        SWAP 275 9 */           ( days in year before this month)
        + +                     ( add DD, days in year and misc)
        S>D 1.721029 D+         ( add Julian date of Jan 1, 0 AD)
        367 R> UM* D+           ( add days of prior years )
        ;
```

The Julian date computed in the above example refers to January 1, 1950 as day zero. This is because we used a single integer to represent the date, and it is limited to a range from -32768 to 32767, enough to cover about 89 years. The true Julian calendar starts at January 1, 4713 BC, which is thought to be the beginning of the world. To represent Julian dates of this large range, we have to use double integers, which have a range from -4,294,967,295 to 2,294,967,294—quite enough for Julian dates.

The following program is adapted to F-PC from Ron Geere's *Forth: The Next Step* (Addison-Wesley, 1986). The original algorithm was published in *Astrophysical J. Suppl.* v41/3, Nov. 1979.

Several instructions dealing with double integers have not been discussed before this example. They are:

`S>D ( n -- d )`
Extend an integer to a double integer.

`D+ ( d1 d2 -- dSum )`
Add two double integers.

`UM* ( n1 n2 -- dProduct )`
Multiple two integers and return a double integer product.

Forth has another stack reserved for subroutine calls and returns. It is called the *return stack*. The function of the return stack is normally hidden from the user, and the user does not have to be concerned with it. However, the return stack is a convenient place to store numbers that are on the data stack, temporarily moving them out of the way. The Forth instructions for moving numbers between the data stack and the return stack are:

`>R ( n -- )`
Pop the top of data stack and push it on the return stack.

`R> ( -- n )`
Pop the top of return stack and push it on the data stack.

`R@ ( -- n )`
Copy the top of return stack and push it on the data stack.

The Game of Life

```
35 CONSTANT white                    ( ASCII # )
32 CONSTANT black                    ( blank )

: address1 ( n -- addr )             ( first area of life objects)
        2047 AND                     ( n modulo 2048 )
        PAD +                        ( add offset to PAD )
        ;

: address2 ( n -- addr )             ( 2nd area for next generation)
        address1                     ( add 2048 to the address1)
        2048 +                       ( let address1 do the modulus)
        ;

: neighbors ( -- )
        2048 0                        ( scan the entire map )
        DO      I 1 + address1 c@   ( add objects in 8 neighbors)
                I 1 - address1 c@ +
                I 79 + address1 c@ +
                I 79 - address1 c@ +
                I 80 + address1 c@ +
                I 80 - address1 c@ +
                I 81 + address1 c@ +
                I 81 - address1 c@ +
                I address1 c@         ( object in this location?)
                IF      DUP 2 =       ( yes.  2 or 3 neighbors? )
                        SWAP 3 = OR
                        IF 1 ELSE 0 THEN ( over-crowded. )
                ELSE    3 =                ( empty location )
                        IF 1 ELSE 0 THEN
                          ( give birth if 3 life neighbors)
                THEN
                I address2 c!   ( store next generation )
        LOOP
        ;

: refresh ( -- )                ( copy next generation to current)
        PAD 2048 + PAD 2048 CMOVE
        ;

: display ( -- )              ( show current map on screen )
        0 0 AT                ( move cursor to upper-left corner)
        PAD 1920 0            ( scan 1920 locations )
        DO      DUP C@        ( life object here? )
                IF WHITE ELSE BLACK THEN  ( show it )
                EMIT
                1 +                       ( next location )
        LOOP    DROP
        ;

: init-map ( addr -- )         ( generate a map from some memory)
        2048 0                 ( look at a 2048-byte area )
        DO      DUP C@ 1 AND   ( use its least significant bit )
                IF 1 ELSE 0 THEN      ( to assign life object )
                I address1 c!         ( in our current map )
                1 +
        LOOP    DROP
        ;
```

Example Two
The Game of Life

The Game of Life is an interesting computer program which simulates the growth and decay of colonies of objects in the memory of a computer. It assumes a memory area is arranged as a two-dimensional map. Each memory location can be either empty or occupied by a life object. Whether a location has a life object or not depends on the life objects in the neighboring locations. The rules of the Game of Life are:

1. A new life object is born in a location if the eight neighbors contain three life objects.
2. A life object dies if the eight neighbors contain fewer than two or more than three life objects.

A life object dies because of loneliness (fewer than two living neighbors) or of overcrowding (more than three living neighbors). A new life is born when there are three living neighbors (presumably a father, a mother, and a priest).

Let's use the computer screen as the map, which contains 25x80 locations. A # represents a life object in a location, and a blank space indicates that there is no life at a location. In memory, we allocate 2048 bytes for a map, of which only 1920 locations are displayed on the screen. The memory area is wrapped in a toroid shape, in which any 80 consecutive bytes are adjacent to the next 80 consecutive bytes; and the first 80 bytes are adjacent to the last 80 bytes. Thus, the locations neighboring location N are N+1, N-1, N+79, N-79, N+80, N-80, N+81, and N-81, and all the numbers are modulo 2048. The modulus of 2048 can be obtained simply by ANDing a number

```
: generations ( n -- )                    ( repeat n generations )
        0 DO    neighbors                 ( compute next generation)
                refresh                   ( copy to current map )
                display                   ( and show it )
        LOOP
        ;

slow                                      ( display every character)
statoff                                   ( hide the headers )
500 init-map                              ( initialize the map )
10 generations                            ( do 10 generations )
```

New Forth instructions introduced in this example are the following:

`CONSTANT ( n -- )`
Define a new instruction which returns n when executed.

`C@ ( addr -- char)`
Fetch a byte from a memory location.

`C! ( char addr --)`
Store a byte to a memory location.

`CMOVE ( a1 a2 n -- )`
Copy n bytes from memory a1 to memory a2.

`PAD ( -- addr )`
Return the address of a free memory buffer.

`AND ( n1 n2 -- n3 )`
Bitwise AND of two 16-bit numbers.

`OR ( n1 n2 -- n3 )`
Bitwise OR of two 16-bit numbers.

with 2047.

The memory area will be assigned as 2048 bytes from PAD, which points to free memory the user can use. Actually, we need two 2048 byte areas, the first to store the map of life objects, and the second to store the next generation of life objects. After computing one generation, the map in the second area is copied to the first area.

Exercise

Think of your favorite board game, and consider the possibility of computerizing it. Maybe it is very difficult to adapt the entire game, but you should be able to find ways to partially computerize it. Consider ways that a computer can help you to improve your playing skill.

Dr. C.H. Ting is a noted Forth authority who has made many significant contributions to Forth and the Forth Interest Group. His tutorial series will continue in succeeding issues of *Forth Dimensions*.

Advertisers Index

Reader Profile:

Mark K. Malmros

My interest in Forth and SBC's (as embedded controllers) is ad vocational. If by chance I find an application of a vocational nature, it is a result of my entrepreneurial inclinations. By background training, I am a physicist (B.S.) and a biochemist (M.S.). Professionally, I have been involved with development of biomedical systems, (immuno-) chemical as well as related instrumentation—ultimately for medical diagnostic applications. So I quite frequently have become involved with micros at various levels during the course of my career. However, I have never seriously needed to develop programming skills. I have, in the past, developed small programs in BASIC, for personal development, if nothing else. BASIC was never a comfortable language: easy, but awkward.

My introduction to Forth came through a need to develop scientific data acquisition capability, PC based, for a special device R&D project (SBIR government funded). ASYST was evaluated and procured for this purpose very successfully. A subsequent employee of mine recognized the similarity to Forth in ASYST's structure (syntax, etc.), and encouraged me to become familiar with Forth. I had found ASYST to be quite intuitive in terms of my needs and, subsequently, I found Forth to be likewise comfortable. It is a logically structured approach to programming and, while I have no other real basis of comparison, just looking at code in C suggests that it is not at all my style. However, my needs for programming are, to repeat, ad vocational at this time. I enjoy working with it to solve some interesting, fun projects.

My interest in observational astronomy (variable stars, etc.) led me to develop a CCD system (charged coupled device) for a 10" Newtonian reflector of my own construction (including mirror), which has kept me interested in Forth as a means to an end—an intellectually pleasant means. I picked up a quality CCD chip, machined the housing, mounting, thermoelectric, and cooling-circulator systems, etc... For my CCD's clock requirements, I found the Inner Access Super 8 (Zilog) Forth SBC running at 20 MHz to be perfect. To obtain the clocking pulses at the speed I desired required writing much of the code in assembly. It worked perfectly and was embedded in an ASYST program (to be converted to a similar program in F-PC) that used DMA with expanded memory to handle the 12-bit pixel data and to manipulate and display the CCD in a quantitative, "photometric" manner. The whole project came to a screeching halt when one of the analog switch drivers for the CCD failed, destroying the CCD; fortunately, I was given a second for development and testing! Next, I discovered that the analog switches were no longer available—anywhere! Given sufficient discouragement, I also found that my professional responsibilities required full focus. I have managed to use the Super 8 to build a nice drive controller for my telescope (stepper-motor controller with many added features), something that would be required once the CCD becomes "astronomically" operational.

...then I will feel that Forth is mine, that I really understand it.

Since that somewhat catastrophic event, my issues of *FD* generally went unread, or were quickly scanned at best. Until I spotted Russell L. Harris' column: I am looking into a project that will require an SBC controller and I would like to complete the CCD project. More importantly, as a kid I always had a screwdriver in my pocket (I am also a Ham: KC3YX) and as with most everything else, I have this unaccountable need to get further into Forth than my various projects have required so far.

The fact that Forth is extensible, self compiling, and readily portable makes it interesting in its own right. Metacompiling, cross-compiling, and cross-assembling have to be more fully understood (by me). While I cannot envision making a living knowing this stuff, I am interested in understanding it at a practical level.

I hope to learn how this is practically accomplished, e.g., by writing an assembler, cross-compiling to an "image" memory, then transporting the cross-compiled code into the target RAM or ROM and booting the device, seeing that it works! Then I will feel that Forth is mine, that I really understand it.

For my CCD project, re-designing the drivers is a big job in itself. And costly. I intend to use eight-bit "flash" D/As so that I can use other CCDs (different clock sequences and voltage levels), fully adaptable by changing the source code in Forth for the Super 8. I will look to the MAX-Forth 68HC11 to take over for my stepper-motor controller system, or some similar ready-made SBC. But I still would like to feel I could "roll my own."

I suggest that I am not representative of the Forth community. For my part, I have found Forth to be compatible with the way I think (at least in part), easy to learn and apply for various real applications. I enjoy working with Forth. I probably will not delve into any other language without a very pressing purpose.

As a result of Mr. Harris' article, I signed onto GEnie (drop me a note at the GE Mail address M.MALMROS) and have browsed the Forth Interest Group's RoundTable. It has lots of interesting files that may help me to learn what I desire, along with the continuing series "On the Back Burner" in *Forth Dimensions*.

Forth Projects

Embedded applications
- CCD clock driver, universal
- telescope drive controller (stepper motor)

Standalone SBCs
- RF plasma chamber controller (proprietary)—vacuum pumping, gas flow, and vac cycling parameters, RF power, and current analyzer

Blue-sky
- anti-carjacking security algorithms
- pocket baseball score keeper (for Little League and high school coaches) that downloads to a PC for full stats, box scores, etc.

A Forum for Exploring Forth Issues and Promoting Forth

Fast FORTHward

Mike Elola

San Jose, California

Hardware as a Creative Medium

Forth programmers often cast their work in the medium of electronic devices such as microprocessors, input and output devices, and read-only memory. The larger software engineering community often takes a dim view of hardware and firmware. Perhaps their discomfort (and mine) arises because of our not feeling comfortable apart from the familiar world of variables, routines, file formats, libraries, and protocols.

Artists who can express themselves in terms of oils, clays, metals, stone, and other media feel increasing rewards with each medium they master. What might be an impossible expression in one medium often can be realized in another, or in a combination of two. I suspect that embedded system programmers feel levels of accomplishment and freedom some of us will never experience.

Embedded systems reveal Forth at its best. Although it was unintentional, the C guru P.J. Plauger instilled this idea in me as I read his "State of the Art" column for *Embedded Systems Programming* (January 1993). "You face your worst problems when writing warm restarts in a high level language such as C." After explaining how embedded systems need to be resilient and recover from errors, Plauger noted that C programmers need a much more detailed understanding of the C run time to create the best embedded systems programs.

Because of his long association with the C language, Plauger probably has source code for various implementations of C, and he regularly offers source code for the standard C libraries. For other C programmers, however, he realizes that C can become a considerable obstacle due to ignorance about the C run-time environment:

"If you write in C (a common choice), you have to know what's going on under the hood.

"...The situation can get worse if you use one or more third-party libraries. In this case, you might not have access to the source code."

Forth does not penalize embedded system programmers this way, because its run-time environment is not hidden from the programmer. Most professional Forth programmers understand the internal operation of the language—as attested to by the fact that they frequently write their own Forth system for each processor they use. This is rarely the case for embedded systems programmers who use other programming languages, for which the task of regenerating the language would require an exhaustive effort.

ANS Forth will make it easier to write warm restarts with its CATCH and THROW additions to Forth. However, imaginative Forth programmers have added many similar routines to a variety of Forths. There is no need to wait for a compiler writer to offer you a Forth with exactly what you need. With Forth, your limits are established by your own imagination, not your lack of access to—or lack of understanding of—Forth's compiling and run-time systems.

(The absence of any hardships upon those who would use a Forth system that they did not generate themselves is a rallying point for Forth programmers—as I reported in the last "Fast Forthward" installment. In that recapitulation of submitted opinions, one of your greatest concerns, and greatest Forth likes, is the ability to understand the internal operation of Forth, as well as any Forth libraries you decide to use.)

Claims of unparalleled flexibility are fair claims for Forth, because Forth systems include routines that invite programmer manipulation of the return stack and all the other run-time resources. So routines such as CATCH and THROW are not only for the language provider to create. As a Forth programmer, you can create such routines as easily as can the Forth compiler writer. In this way, you can extend Forth to fulfill your needs as seen by you (not some distant and unknown compiler writer).

For example, many Forth routines are available to manipulate the contents of data structures that reside in memory. In Forth, these operations require as input parameters real addresses that you are able to pick previously. In most other languages, operations that manipulate memory contents can only be passed "symbol names," because the compiler must be allowed to establish the actual addresses of data. In Forth, you write routines that deal directly with real memory (and directly with real hardware). Thereby you are truly empowered to create and reshape Forth at a meaningful level. In terms of the freedom it provides, Forth resembles an assembly language—which is as close to direct manipulation of hardware as we can get. The close control of actual hardware is ultimately what embedded systems programmers need in order to minimize costs. (Applications developed in Forth are typically more compact than those written in assembly language, while it still affords the level of control needed.)

Forth strikes out in an unusual direction as far as programming languages are concerned. Its goal is to offer the programmer the needed power over its own run-time and development environments. This becomes a significant advantage, particularly in application areas where an understanding of what's "under the hood" is vital.

The approach taken by other languages is to deny access to critical processor (run-time) resources such as a return stack. This is done in the hope of removing the possibility of disruptive actions that a programmer might instigate. In those languages, the reasoning is that a constrained programmer is a better programmer. In contrast, the Forth approach lets you create routines that extend or reshape the run-time system to obtain the support you need, such as a new form of warm restart.

If C has nuisances for the embedded systems programmer to overcome, C++ offers no relief either, according to Plauger. After explaining how C++ makes programming warm restarts easier, Plauger goes on to lament the penalty of too much overhead processing in C++: "If the penalty is as high as it seems to be, I hope a better way exists. So far, however, the results have been daunting."

Because of his long association with C, our chances of making a Forth convert of Plauger are not very good. However, he has opened a topic of discussion in which Forth programmers should participate. So open your copies of *Embedded Systems Programming*, read the column, and send him your comments. At the same time, send a copy to the FIG office so that you can help contribute to discussions in *Forth Dimensions* too.

I suspect that Plauger has arrived at one of those little-acknowledged areas that can help explain Forth's appeal to many programmers, particularly embedded systems programmers. Let's hear your take on this subject.

Recognition of ESPs

When the ranks of FIG reached its peak at 4800 members back in 1984, the embedded systems marketplace was only a glimmer of what it is today.

Numerous microprocessors, microcontrollers, and associated components are an important part of our world now. With this technology, businesses both large and small have created many sought-after products. Some of these products are experimental ones that help advance our scientific knowledge. Many of these products are merely practical ones.

Forth has proven to be a good match for a wide variety of products. On the scientific side, Forth has flown as part of the equipment aboard space shuttle missions. As part of a more down-to-earth device, Forth is being carried over much of the planet inside the palmtop computers used by Federal Express mail carriers.

Given these developments over the past ten years, embedded systems programmers must represent an ever-increasing proportion of the FIG membership. How much have embedded system programmers contributed to the influx of new FIG members? To help us chart an appropriate course for FIG, we should at least gather information about about the present composition of FIG.

With a stamp, an envelope, and your business card, you can help provide the information FIG needs. From your business cards, we should be able to determine how many of you are currently involved in embedded systems work, as well as other fields of endeavor. If you work tends to vary, send us one copy each of the different business cards you regularly use.

To provide even more information for FIG, consider writing a keyword or two on the back of your card(s). You may choose from the following terms if you wish: industrial control, avionics, aerospace, communications, instrumentation, vending machines, computer peripherals, robotics, medical, business, or test equipment.

Product Watch

FEBRUARY 1993

MicroProcessor Engineering Ltd. announced the MPE RTX2001 PowerBoard based on the Harris RTX family of processors. The PowerBoard follows the Eurocard (220x100mm) form factor. Due to a fully deterministic interrupt latency of 400 nanoseconds, this system supports data aquisition rates as high as one megaword per second according to MPE. Because of the RTX processor's Forth-oriented architecture, including dual on-chip stacks, it executes 10 million (Forth) instructions per second. The MPE optimizing cross-compiler (and XShell interpreter) runs on a PC and optimizes code to suit the RTX processor's ability to overlap the execution of up to five Forth instructions. The RTX PowerBoard is available with either an RTX-2000 or RTX-2001A (lacks a single-instruction multiply), and includes a DIN edge connector, 64K of zero-wait RAM (expandable to 512K along with 512K of EPROM), RS-232 and RS-485 ports, and 8255 and 8254 components for 24 digital I/O lines plus several timers. A PC Adaptor is available for a standard PC/ISA backplane. Other adapters are available for use with STE and VME buses.

MARCH 1993

AM Research announced the amr451LC and amr552LC. They are low cost, low power (all CMOS) systems based on the 80C451 and 80C552. Like other members of this product line, these systems simulate most members of the 8051/8052 family at speeds up to 30MHz. FIG members are offered their first 'System' purchase at a 25% discount. AM Research 'Systems' come with an extensive manual, a Forth for your PC, a Forth for the target, the target, editor, assembler, communications support, disassembler, decompiler, cables, and a 100mA power supply brick to fuel the target hardware. At prices starting at $100, the AM Research arm451LC and amr552LC hardware includes a large prototyping area within a 4x6" PCB, RS-232, lithium battery, +5V regulator and DC input jack, CMOS RAM and ROM sockets, and an optional stacking bus connector. Options include eight-channel 8-, 10-, or 12-bit A/D input (up to 176 I/O lines), 2 line by 40 character LCD, and 16-button keypad.

Companies Mentioned

AM Research
4600 Hidden Oaks Lane
Loomis, CA 95650
Phone: 916-652-7472

MicroProcessor Engineering, Ltd.
133 Hill Lane
Southampton SO1 5AF
Fax: 0303 339691
Telex: 474695 FORMAN G
Phone: 0703 631441

```
\               ****************************************
\               * 32-bit Floating-Point Input and Output *
\               ****************************************
\ Numbers to be floated must include a decimal point when entered.
\ DPL contains the number of digits entered after the decimal point.
: FLOAT  ( n -- f)              \ float the last entered number.
  dpl @ negate trim
;
: F.  ( f --)                   \ print a floating number in fixed format.
  >r dup abs 0
  <# r@ 0 max 0 ?do ascii 0 hold loop
  r@ 0<
  if r@ negate 0 max 0 ?do # loop ascii . hold
  then r> drop #s rot sign
  #> type space
;
\               ************************************************************
\               *     32-bit Floating-Point Transcendental Functions       *
\               *     Taken from Zen slide rule by Nathaniel Grossman      *
\               ************************************************************
comment:
These functions are calculated to an accuracy of about one or two units in the
third decimal place.  Domains of the functions - scale input if need be to
keep within range.  The CORDIC algorithm is used, so these routines are not fast.
SQRT                     0.03   -    2.42
LN                       0.10   -    9.58
SIN,COS,TAN             -1.74   -    1.74
ATAN                 -infinity - +infinity
SINH,COSH,TANH           -1.13   -    1.13
ATANH                    -0.81   -    0.81
comment;
: F2*   2 0  f*   ;                             \ alternate?         2dup f+
: F/2 swap s>d 2dup d2* d2* d+ rot 1- trim ;    \ mantissa * 5, dec exp by 1
: F/2^N   ( r n --- r/2^n)   0 ?do f/2 loop   ; \ divide by 2, n times

\ Convenient renaming of existing words.  Either these or original names work.
' 2DROP alias FDROP
' 2DUP alias  FDUP
' 2OVER alias FOVER
' 2SWAP alias FSWAP
' 2ROT alias  FROT
' 2!   alias  F!
' 2@    alias F@
' 2VARIABLE alias FVARIABLE

\ Extra floating point words
: F0< DROP 0< ;
: F, FLOAT , , ;
: FCONSTANT FLOAT 2CONSTANT ;

\ Constants, Variables, Deferred Words, and Arrays
4       CONSTANT F#BYTES                 VARIABLE MODE_FLAG
0.6073 FCONSTANT 1/K                     VARIABLE DELTA_FLAG
1.2076 FCONSTANT 1/K'                    VARIABLE NDX
0.0000 FCONSTANT F0                     FVARIABLE F-BIN
1.0000 FCONSTANT F1                     FVARIABLE XX
0.2500 FCONSTANT F1/4                   FVARIABLE YY
                                        FVARIABLE ZZ
DEFER EPS                               \ it will be either +EPS OR -EPS
DEFER !STACK                            \ it will be R-STORE or V-STORE
DEFER DO-IT                             \ it will be either ROT'ING or VEC'ING
: FARRAY                                \ array-building word
   CREATE                               \ ( -- ) compile time stack
   DOES>                                \ ( n --- adr ) run time stack
   swap f#bytes * + f@ ;                \ calc address of nth entry
```

```
FARRAY  +EPS     .7854    F, .4636    F, .2450    F, .1244    F, .06242    F,
                 .03124   F, .05162   F, .007812  F, .003906  F, .001953   F,
                 .0009766 F, .0004883 F, .0002441 F, .0001221 F, .00006104 F,
FARRAY  -EPS     .0000    F, .5493    F, .2554    F, .1257    F, .06258    F,
                 .03126   F, .01563   F, .007813  F, .003906  F, .001953   F,
                 .0009766 F, .0004883 F, .0002441 F, .0001221 F, .00006104 F,

: DELTA_SIGN   delta_flag @  if fnegate then ;
: MODE_SIGN    mode_flag @   if fnegate then ;
: R-DELTA=    ( r --- r )     fdup f0< delta_flag ! ;
: V-DELTA=    ( r --- r )     fdup f0< not delta_flag ! ;
: R-STORE     ( x y z --- )   r-delta=  zz f!  yy f!  xx f! ;
: V-STORE     ( x y z --- )   zz f!  v-delta=  yy f!  xx f! ;
: NEW_Z      zz f@ ndx @ eps delta_sign f- ;
: NEW_X      xx f@ yy f@ ndx @ f/2^n delta_sign mode_sign f+ ;
: NEW_Y      yy f@ xx f@ ndx @ f/2^n delta_sign f+ ;
: ROT'ING    new_x new_y new_z  r-store ;
: VEC'ING    new_x new_y new_z  v-store ;
: MODE=+1    -1 mode_flag ! ['] +eps is eps ;
: MODE=-1    0 mode_flag !  ['] -eps is eps ;
: CORDIC ( xstart ystart zstart --- xend yend zend )
  !stack mode_flag @ dup 0= >r
  if 0 ndx ! do-it then
  4 1 do i ndx ! do-it loop
  r@ if 4 ndx ! do-it then
  14 4 do i ndx ! do-it loop
  r> if 13 ndx ! do-it then
  14 ndx ! do-it xx f@ yy f@ zz f@
;
: FCOS&SIN ( r --- cos{r}  sin{r}  f )
  1/k fswap f0 fswap ['] r-store is !stack
  mode=+1 ['] rot'ing is do-it cordic
;
: FCOS    ( r --- cos{r})    fcos&sin fdrop fdrop ;
: FSIN    ( r --- sin{r})    fcos&sin fdrop fswap fdrop ;
: FTAN    ( r --- tan{r})    fcos&sin fdrop fswap f/    ;
: FCOSH&SINH ( r --- cosh{r} sinh{r} f )
  1/k' fswap f0 fswap ['] r-store is !stack
  mode=-1 ['] rot'ing is do-it cordic
;
: FCOSH    ( r --- cosh{r})    fcosh&sinh       fdrop fdrop ;
: FSINH    ( r --- sinh{r})    fcosh&sinh fdrop fswap fdrop ;
: FTANH    ( r --- tanh{r})    fcosh&sinh fdrop fswap f/    ;
: FALN     ( r ---  exp{r})    fcosh&sinh       fdrop f+    ;

: FLN ( r --- ln{r} )
  fdup f1   f+ fswap f1   f- f0  ['] v-store is !stack
  mode=-1 ['] vec'ing is do-it   cordic
  fswap fdrop fswap fdrop f2*
;
: FSQRT  ( r --- sqrt{r} )
  fdup f1/4 f+ fswap f1/4 f- f0  ['] v-store is !stack
  mode=-1 ['] vec'ing is do-it   cordic
  fdrop fdrop 1/k' f*
;
: R>P ( x y --- {x^2 + y^2}^1/2  arctan{y/x} )
  1/k f* fswap 1/k f* fswap f0   ['] v-store is !stack
  mode=+1   ['] vec'ing is do-it   cordic  fswap fdrop
;
: P>R ( radius angle --- x y )
  fover fswap fcos&sin fdrop frot f* frot frot f* fswap
;
: FATAN    ( r --- arctan{r})    f1 fswap r>p  fswap fdrop ;
: FATANH   ( r --- argtanh{r})  f1 fswap f0  mode=-1
  ['] vec'ing is do-it  cordic  fswap fdrop fswap fdrop ;
```

("Back Burner," continued from page 43.)

the index of the vocabulary, but also additional vocabularies, if any, to be searched. (The four hexadecimal digits specifying the search order are read either right to left or left to right, depending upon the implementation.) Thus, FORTH is identified by the index 1, ASSEMBLER by the index 3, and EDITOR by the index 5. The value 0001 specifies that an unsuccessful search of the vocabulary FORTH will terminate, whereas the values 0013 and 0015 specify that a non-successful search in either ASSEMBLER or EDITOR, respectively, will be resumed in FORTH.

User variable CONTEXT contains a 16-bit value (16 bits equate to four hexadecimal digits) which specifies the vocabulary search order. The first vocabulary in the search sequence is termed the *primary* vocabulary. User variable CURRENT contains a 16-bit value (again, four hexadecimal digits) which specifies the vocabulary into which the next definition is to be compiled (compilation is into the primary vocabulary). As you may have surmised, in each case the 16-bit values loaded to CONTEXT and CURRENT are the values associated with vocabulary names. CONTEXT is set by execution of a vocabulary name, whereas CURRENT is set by the word DEFINITIONS, which simply copies CONTEXT into CURRENT. DEFINITIONS typically appears immediately following a vocabulary name, as in the phrase ASSEMBLER DEFINITIONS.

Off With Their Heads!

In general, a dictionary search occurs only in the process of defining a word in terms of words previously defined, and in the attempt to execute a word which has been interpreted by the outer interpreter (the *text* interpreter). Many applications—particularly embedded systems—*neither define nor interpret*, and therefore never conduct a search. Conversely, searches are commonplace in the development environment.

A dictionary entry consists of a *head*, which is optional, and a *body*. The head is the repository for the information which allows the dictionary to be searched; it typically is composed of the following elements:

- a *locate field*, which is optional; the field contains either the number of the source block which was loaded to compile the entry, or zero, if the word was defined at the keyboard
- a *link field*, which contains a pointer to the link field of the previous word in the dictionary thread into which the entry is linked
- a *name field*, the first byte of which contains the character count of the name prior to truncation, followed by either the full name or only the first three characters of the name

Although the majority of words in the dictionary of the development system have heads, words in the *nucleus* which are never directly referenced may be *headless*, thus minimizing the size of the nucleus. The ability to have headless code in the nucleus is a consequence of the fact that the nucleus is the product of metacompilation. Whether one is creating an application or a new development environment, metacompilation allows the compilation of headless code. This capability can be vital to the economic success of an embedded system, since the memory requirement is often the predominant consideration in specification and configuration of the hardware.

To Each His Own

In metacompilation, we have need of several vocabularies within the dictionary of the development system. First of all, we need the normal vocabularies FORTH and EDITOR. Since we typically have no need to define additional code words which execute on the development system, we may dispense with the normal ASSEMBLER vocabulary. We place words belonging to the metacompiler in a vocabulary named COMPILER. Likewise, we place in a *new* vocabulary named ASSEMBLER words belonging to the assembler of the metacompiler (which we might term the *meta-assembler*). Note the consequence of reusing the name ASSEMBLER: once the metacompiler is loaded, words in the *normal* ASSEMBLER vocabulary become inaccessible.

All the vocabularies enumerated in the preceding paragraph are normal Forth vocabularies, in the sense that they are accessed via CONTEXT and CURRENT, they may be searched by the normal version of -', and words may be compiled into each of them by the normal Forth compiler. The vocabulary FORTH is redefined, for the purpose of redefining the search order; additionally, FORTH is made immediate. The vocabulary definitions are as follows:

```
HEX
0071 VOCABULARY FORTH   IMMEDIATE
0017 VOCABULARY COMPILER  IMMEDIATE
0173 VOCABULARY ASSEMBLER
```

Note that the search order of COMPILER includes FORTH, while that of ASSEMBLER includes both COMPILER and FORTH.

We require yet the services of one to three additional vocabularies. The need arises from the fact that compilation and dictionary searches are inseparably united. In the compilation of a high-level word, one must search the dictionary in order to obtain the compilation addresses of the component words. In the compilation of a code word, the dictionary must be searched in order to find and execute the assembler opcodes and directives.

In metacompilation, the addresses required are those of *the future application system*. The physical address at which the application code is compiled within the development system differs from the physical address at which the code will reside within the application system. More importantly, code compiled into the application dictionary may have heads, may be entirely headless, or may be a mixture of headed and headless entries.

If every application word has a head, one might consider searching the application dictionary. However, if the application dictionary is compiled to disk, virtual memory operators would be required, and search time would become a major impediment to the compilation. Conversely, if the compiled application is entirely headless, no search is

possible. These and other considerations lead us to conclude that searches of the application dictionary in an attempt to obtain the required compilation addresses are, generally, impractical, and, typically, impossible.

Enter Knight, white as a ghost...

Charging to our rescue is the aforementioned set of additional vocabularies. We shall require, in the development environment, an *application vocabulary* for each category of application words. If the application is itself a development environment, the application vocabularies will be (APPLICATION FORTH), (APPLICATION ASSEMBLER), and (APPLICATION EDITOR). Headless applications have need of only one vocabulary, (APPLICATION FORTH). I have placed the names in parentheses to emphasize a curious fact: the additional vocabularies are unnamed. The fact that names are not defined for the application vocabularies has led to the appellations *ghost* and *phantom*; equally applicable are the terms *hidden* and *invisible*.

The invisible or phantom vocabularies eliminate the need to search the application dictionary. For each entry (with or without head) in the application dictionary, a corresponding *confederate* entry is compiled into one of the invisible vocabularies. Confederate entries typically are composed of a normal head, together with a body in which the parameter field consists of nothing more than the application-system address of the corresponding entry in the application dictionary. The typical run-time behaviour of a confederate is to compile an item (e.g., an address) into the application dictionary. The vocabulary structure of the application dictionary is mirrored in the invisible vocabularies. Confederate entries in the invisible vocabularies are compiled at the time the corresponding entries are compiled into the application dictionary.

Seek and Ye Shall Find

Although the invisible vocabularies are not named, there is associated with each a vocabulary index. Indices of the invisible vocabularies are simply the indices of the corresponding development environment vocabularies incremented by the value 0A hexadecimal. Thus, the indices for (APPLICATION FORTH), (APPLICATION ASSEMBLER), and (APPLICATION EDITOR) are, respectively, 000B, 00BD, and 00BF. This choice of indices ensures that the invisible vocabularies can be searched *only* by the metacompiler. However, although the invisible vocabularies are accessed only by the metacompiler, they are implemented no differently than the other vocabularies in the development environment. That is to say, entries in the invisible vocabularies are compiled at the end of the development system dictionary, thus causing the normal dictionary pointer to be incremented; entries in the invisible vocabularies are linked into one of the eight dictionary threads, the thread being selected by hashing with the vocabulary index; etc.

The normal Forth word -' is used by the metacompiler to search the invisible vocabularies, via a surrogate for CONTEXT. (Recall that -' searches the primary vocabulary specified by CONTEXT.) Rather than maintaining a pair of variables corresponding to CONTEXT and CURRENT, the metacompiler utilizes only a single variable, VOC. When the metacompiler invokes -', CONTEXT is loaded with the value in VOC, while the old value of CONTEXT is preserved and afterward restored. VOC is initially set to the value 000B.

The Plot Thickens

The basic operation of the metacompiler, upon parsing a word in the application source code, is to search the application vocabularies for the word and then *execute* the word. As previously noted, the typical run-time behaviour of a word compiled into one of the application vocabularies is to *compile* an item (e.g., an address) into the application dictionary.

In addition to application words, the metacompiler must be able to find words such as compiler directives. Moreover, words such as compiler directives must be *executed* at compilation time, rather than being compiled. A complication arises from the fact that the metacompiler has access only to the application vocabularies.

The solution lies in a word which I (taking my cue from the name IMMEDIATE) hereupon dub NOW. Because the metacompiler *executes* words it finds in the application vocabularies, the only thing NOW must do is move (i.e., relink) words from the normal vocabularies (FORTH, COMPILER, ASSEMBLER) to one of the (unnamed) application vocabularies. As is the case with IMMEDIATE, the word NOW operates on the most recent definition.

A Parting Shot

Over the past year, this column has generated an average of just over one response per month. Although I myself am inclined to conclude that no one is interested in the column, colleagues have suggested that even the *most* interested of readers seldom, if ever, bothers to write; this supposedly has something to do with the independent nature of Forth devotees. I consider reader feedback extremely important, whether it is a hail of brickbats or a shower of roses. I honestly would prefer an avalanche of complaints to a total absence of comment. I want to know the degree of your interest in the subject matter, and whether the level of presentation is appropriate to your needs. If you don't care about metacompilation, what *would* interest you? A 19-cent postcard every couple of months from every serious reader would be most helpful, and would be greatly appreciated. Alternatively, phone me, send me a fax, or leave me a message on GEnie. We authors are a vain and insecure lot, and desperately need to know that there really *is* an audience out there, beyond the glare of the footlights.

R.S.V.P.

Russell Harris is an independent consultant providing engineering, programming, and technical documentation services to a variety of industrial clients. His main interests lie in writing and teaching, and in working with embedded systems in the fields of instrumentation and machine control. He can be reached by phone at 713-461-1618, by facsimile at 713-461-0081, by mail at 8609 Cedardale Dr., Houston, Texas 77055, or on GEnie (address RUSSELL.H).

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*)*
  Call GEnie local node, then
  type M710 or FORTH
  SysOps:
  Dennis Ruffer (D.RUFFER),
  Leonard Morgenstern (NMORGENSTERN),
  Elliot Chapin (ELLIOT.C)

BIX (ByteNet)
For information,
call 800-227-2983

- Forth Conference
  Access BIX via TymNet, then type j forth
  Type FORTH at the : prompt
  SysOp:
  Phil Wasson (PWASSON)

- LMI Conference
  Type LMI at the : prompt
  LMI products
  Host:
  Ray Duncan (RDUNCAN)

CompuServe
For information,
call 800-848-8990

- Creative Solutions Conf.
  Type !Go FORTH
  SysOps: Don Colburn, Zach Zachariah, Ward McFarland, Jon Bryan, Greg Guerin, John Baxter, John Jeppson

- Computer Language Magazine Conference
  Type !Go CLM
  SysOps: Jim Kyle, Jeff Brenton, Chip Rabinowitz, Regina Starr Ridley

Unix BBS's with forth.conf (ForthNet and reachable via SprintNet node casfa on TeleNet.)*

- WELL Forth conference
  Access WELL via CompuserveNet
  or 415-332-6106
  Fairwitness:
  Jack Woehr (jax)

PC Board BBS's devoted to Forth (ForthNet)*

- British Columbia Forth Board
  604-434-5886
  SysOp: Jack Brown

- Grapevine
  501-753-8121 to register
  501-753-6859
  SysOp: Jim Wenzel

- Real-Time Control Forth Board
  303-278-0364
  SprintNet node coden on TeleNet
  SysOp: Jack Woehr

Other Forth-specific BBS's

- Laboratory Microsystems, Inc.
  213-306-3530
  SprintNet node calan on TeleNet
  SysOp: Ray Duncan

- Druma Forth Board
  512-323-2402
  SysOps: S. Suresh, James Martin, Anne Moore

Non-Forth-specific BBS's with extensive Forth libraries

- DataBit
  Alexandria, VA
  703-719-9648
  PCPursuit node dcwas
  SysOp: Ken Flower

- PDS*SIG
  San Jose, CA
  408-270-0250
  SprintNet node casjo

- Programmer's Corner
  Baltimore/Columbia, MD
  301-596-1180 or
  301-995-3744
  SprintNet node dcwas

International Forth BBS's

- Melbourne FIG Chapter
  (03) 809-1787 in Australia
  61-3-809-1787 international
  SysOp: Lance Collins

- Max BBS *(ForthNet*)*
  United Kingdom
  0905 754157
  SysOp: Jon Brooks

- Serveur Forth
  Paris, France
  (1) 41 08 11 75
  300 baud (8N1) or
  1200/75 E71
  or call (1) 41 08 11 11 *for*
  1200–9600 baud (8N1)
  Minitel, higher-speed, & alternate carriers available.
  Sysop: Marc Petremann

- Sky Port *(ForthNet*)*
  United Kingdom
  44-1-294-1006
  SysOp: Andy Brimson

- SweFIG
  Per Alm Sweden
  46-8-71-35751

- NEXUS Servicios de Informacion, S. L.
  Travesera de Dalt, 104-106,
  Entlo. 4–5
  08024 Barcelona, Spain
  + 34 3 2103355 (voice)
  + 34 3 2147262 (modem)
  SysOps: Jesus Consuegra, Juanma Barranquero
  barran@nexus.nsi.es (preferred)
  barran@nsi.es
  barran (on BIX)

**ForthNet is a virtual Forth network that links designated message bases in an attempt to provide greater information distribution to the Forth users served. It is provided courtesy of the SysOps of its various links.*

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 c/o Unisyn
301 Main, penthouse #2
Longmont, CO 80501
303-924-9193

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

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

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

Charles Keane
Performance Pkgs., Inc.
515 Fourth Avenue
Watervleit, 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

On the Back Burner #7

"The King is Dead! Long Live the King!"

Conducted by Russell L. Harris
Houston, Texas

The last column was devoted to basic nomenclature, together with a recommendation as to how you can escape or avoid the dreaded *Lone Ranger syndrome* as you pursue an understanding of Forth. In this episode, we turn our attention to the environment in which the metacompiler operates.

What I shall describe is a system typical of those I have studied, i.e., something of a *generic* polyFORTH. If your system differs somewhat, don't despair. Your time won't be wasted. Once you understand the manner in which one system works, you will find it much easier to understand other Forth systems. The similarities greatly outweigh the differences.

One Stone, Many Birds

A feature of Forth which plays a central role in metacompilation is the support of multiple vocabularies. Vocabularies are basic to Forth and to good Forth programming practice. Vocabularies serve to simplify source code, maintain expressiveness, and segregate code by category.

The first two functions are based on the concept that the meaning of a word is determined by the context in which the word appears. Without vocabularies, we would need a separate word—a *synonym*—for each context in which a Forth word is used. However, not infrequently in the course of our programming activity, we are unable to find a sufficient number of synonyms for a given word. Also, there are instances in which a synonym won't do: at times, a particular word simply *must* appear in more than one context. A possible work-around lies in the use of typography and symbolism. However, in Forth, the primary use of typography and symbolism is to indicate relationships between words. For example, `(TYPE)` is the device-specific code for the word `TYPE`, while `(type)` is the code routine associated with `(TYPE)`. The use of typography and symbolism to differentiate synonyms would interfere with this scheme and cause confusion.

The third function, that of code segregation, is the most important of the three with respect to metacompilation. In the process of metacompilation, application code is compiled to a special *application dictionary*, located either in RAM or on disk. Additionally, several categories of words are compiled within the dictionary of the development system; these categories must be searched while the compilation is in progress. The vocabulary structure allows the programmer to specify the category into which code is to be compiled, the categories to be searched, and the sequence of multiple-category searches.

Learning the Ropes

In Forth, a *dictionary* is a region of memory into which words and data structures are compiled. Recall that *compilation* is simply the process of writing to a dictionary. The process of metacompilation builds a dictionary for a future system, the *application*. Although the application dictionary may be too large to be accommodated within the typical 64 Kbyte Forth environment (the *operating environment*), we can easily build the application dictionary in memory *external* to the operating environment. With development systems such as the IBM-PC, we can write compiling words to directly access RAM outside the 64K operating environment. Alternatively, we can use virtual memory techniques to write compiling words which access disk storage (including so-called "RAM disks"). In this latter case, the application dictionary is termed a *virtual dictionary*.

Two mechanisms are simultaneously employed in order to systematize dictionary entries. From the standpoint of the programmer, the dictionary is compartmentalized into a number of *vocabularies*. From the standpoint of the Forth system, the primary compartmentalization of the dictionary is in the form of several (typically, eight) *dictionary threads*. Each dictionary entry has membership in a single vocabulary, but each entry also is linked into one of the eight threads.

The linking is done on the basis of a *hashing scheme* or *hashing algorithm*. When `-'` searches for a given word, the hashing algorithm combines the corresponding vocabulary index with the first character of the name in order to determine the thread to be searched. Thus, effectively, the entire dictionary may be searched by searching only one-eighth of its entries. Note that there is only an indirect relationship between a vocabulary and a dictionary thread: potentially, each thread links words of *all* vocabularies, and words of any given vocabulary may be found on *each* of the dictionary threads.

Building Your Vocabulary

In a typical development environment, one finds initially at his disposal three vocabularies: `FORTH`, `ASSEMBLER`, and `EDITOR`. Corresponding to each vocabulary is an *index* which ranges from 1 to F hexadecimal, and which may assume only odd values. Vocabularies are defined by phrases of the following sort:

```
HEX
0001 VOCABULARY FORTH
0013 VOCABULARY ASSEMBLER
0015 VOCABULARY EDITOR
```

in which the associated hexadecimal digits specify not only

(Continued on page 40.)

CALL FOR PAPERS

for the fifteenth annual and the 1993

FORML CONFERENCE

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

Following Thanksgiving
November 26–November 28, 1993
Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California U.S.A.

Theme: **Forth Development Environment**

Papers are invited that address relevant issues in the establishment and use of a Forth development environment. Some of the areas and issues that will be looked at consist of networked platform independence, machine independence, kernel independence, Development System/Application System Independence, Human-Machine Interface, Source Management and version control, help facilities, editor development interface, source and object libraries, source block and ASCII text independence, source browsers including editors, tree displays and source data-base, run-time browsers including debuggers and decompilers, networked development-target systems.

Additionally, papers describing successful Forth project case histories are of particular interest. Papers about other Forth topics are also welcome.

Mail abstracts of approximately 100 words by September 1, 1993.
Completed papers are due November 1, 1993.

We anticipate a full conference this year.
✔ **Priority will be given to participants who submit papers.**
✔✔ **Earlier papers will be given preference in choosing longer presentation periods**

John Hall, Conference Chairman

Robert Reiling, Conference Director

Information may be obtained by phone or fax from the
Forth Interest Group, P.O. Box 2154, Oakland, CA 94621. 510-893-6784, fax 510-535-1295
This conference is sponsored by FORML, an activity of the Forth Interest Group, Inc. (FIG).

The Asilomar Conference Center combines excellent meeting and comfortable living accommodations with secluded forests on a Pacific Ocean beach. Registration includes use of conference facilities, deluxe rooms, all meals, and nightly wine and cheese parties.