ISSN 0265-5195

Forthwrite FIGUK

Issue 106 April 2000

Editorial

I'm sorry this issue is late - my fault, not your contributors - but I'm sure it was worth the wait.

A new development to report - you can now download Forthwrite from the web-site. You don't need to lend you precious copies to friends and colleagues any more. This is an experiment, so please tell us what you think. We do not see any threat to the printed copy which will always remain one issue ahead of the electronic version.

IRC has been flourishing, with Forthers from France and The Netherlands joining our regulars from Germany. Do give it a try.

Dave Pochin's web-site has recorded 5,000 hits for his Getting Started guide - an effective measure of Forth's popularity.

Welcome to some more new members - Christian Hellon in Bradford and Arthur Berg in Aberdeen.

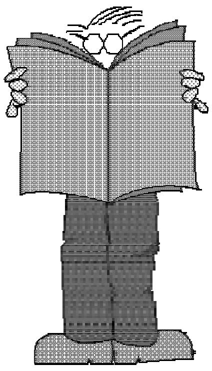
News of two projects from recently-joined members. Matt Gumbley (mgumbley@enigmadata.co.uk) is working with 2 others on an hForth port to the Psion Series 5. Jan Coombs is a digital designer who is experimenting with "digit serial" processors.

Please get in touch if you don't receive monthly e-mail from me (IRC reminders, job offers etc.) as it means I don't know your current e-mail address and you're missing out.

Until next time, keep on Forthing,

[illegible]

Dave Abrahams
0161 477 2315
d.j.abrahams@cwcom.net



Forth News

Web Sites

How many people know that there is a Chat Room for Forth in ICQ?

This was first announced in Forth News in June 99.

Computer Solutions (Comsol) have added a new resource to their web site for all designers of embedded micro-processors. This is intended to have a European (i.e. non-USA) bias and provide a suitable starting point for locating resources.

http://www.computer-solutions.co.uk

Ulrich Hoffman has set up an experimental "Wiki" web site for the exchange of ideas about Forth.

Wiki Web Servers allow everyone to change pages and so their content will grow in a collaborative effort.

http://hammer.prohosting.com/~uho/wiki/html/Forth/FrontPage.htm

International FIG have announced a web-based search service for Forth practitioners. Visitors can enter keywords and retrieve details of FIG members operating in that field. International FIG members must remember to enter their own details - it won't be done for you.

http://www.forth.org/forthexperts.html

If you are looking for Forth expertise in the UK, see *http://forth.org.uk* for a list of training services and contact FIG UK to pass on your requirements to FIG UK members who have registered their availability.

Forth Applications and Utilities

The Open Terminal Architecture project undertaken by Forth Inc and MPE Ltd for Europay

International "is in increasing use in Europe. The technology itself has just become an **ISO standard**". Europay is Europe's largest financial services company.

Elizabeth Rather reports that the hand-held package tracking devices used by Federal Express have been programmed in Forth since 1986 and Forth Inc. are currently working on the **3rd generation design**.

Database access via **SQL** is not a common feature in Forth. Here is sample code to access any ODBC database.

ftp://forthcad.com/pub/web682c3/forth/

files: *ODBC-DB.F*

ODBC-TEST.F

WinCV is a Windows shareware utility written in Win32Forth. It is a File Manager and Picture Viewer.

http://netcity.hinet.net/lccw/wincv023.exe

A parser that will translate **Forth into C** can be found at:

http://www.complang.tuwien.ac.at/projects/forth.html

Projects

To join a project to build a Forth-based **operating system**, visit ...

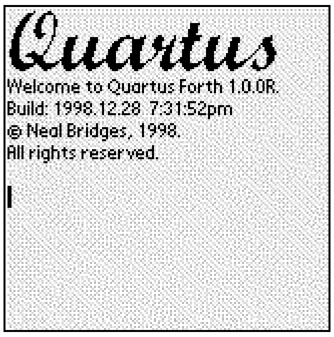
http://www.onelist.com/community/fcvm

VHDL is a language for building chips such as small micro-processors. Don Golding has started a design for a **Forth processor** and is looking for assistance.

dgolding@angelusresearch.com

Free Systems

Quartus Forth for the Palm Pilot runs on Unix, Windows,



Welcome to Quartus Forth 1.0.0R.
Build: 1998.12.28 7:31:52pm
© Neal Bridges, 1998.
All rights reserved.

and the Mac OS using the **Palm OS Emulator** now available for all three platforms.

http://www.quartus.net/

DELTA Forth 1.0 is now available. This is the latest in a series of updates for this **Java-based Forth**. Help in developing it further is welcome.

DELTA Forth was rated 5 stars at SuperShareware (*www.supershareware.com*) and Top 25% at JARS (*www.jars.com*)

You may download your free copy from:

http://www.dataman.ro/dforth

A new release of Ralph Hempels pbForth for **Lego Mindstorms** is now available. Download version 1.1.1 from:

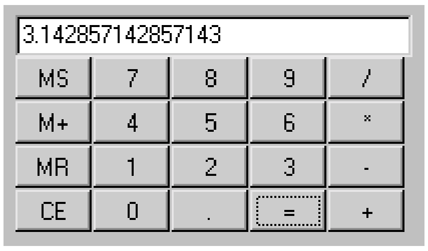
http://www.hempeldesigngroup.com/lego/pbFORTH

Wonyong Koh's hForth is a well-respected ANS Forth for a variety of processors. Now available to suit **ZMASM for Z80**.

http://www2.whidbey.net/

~beattidp/

HolonJForth gives a new twist on mixing languages and enables you to write in Forth,



use Java libraries and compile to run on a **Java Virtual Machine** with no performance penalty.

http://holonforth.com/tools/java/

holonj.htm

Rick Hohensee has made Linux available on a **single floppy** with 2 Forths. Look forward to hearing more about this technology.

ftp://linux01.gwdg.de/pub/

cLIeNUX/interim

eForth is now available direct from the web site of Bill Muench, its author.

This version includes much material missing from the previous 1997 version.

http://members.aol.com/forth/

Quartus Forth, well-known for Palm Pilot, is available for the **Royal da Vinci** PDA. It was

previewed in the Nov. issue of

Forth News and is now available from

http://www.royal.com/davinci/

The free evaluation version can be found at:

http://www.quartus.net/files

Commercial Systems

iForth is a high-performance 32-bit ANS Forth available for several platforms including Linux and Windows NT. Full details including graphics output are on the web site.

http://www.iae.nl/users/mhx/

support.html

iForth costs 200 Guilders (currently about £56).

Have a look at this page on the same site for 6 examples of **classic programs** in ANS Forth.

www.iae.nl/users/mhx/

programs.html

Is **VFX the fastest Forth** yet? Published benchmarks give 4 - 10 times faster than nearest commercial rival and within 25% of hand-crafted assembler

You can now try out MPE's performance claims for free by downloading the very latest ProForth for Windows with the VFX code generator:

www.mpeltd.demon.co.uk

Hardware

Elizabeth Rather reports on comp.lang.forth that the **RTX2000** is only available in the RAD-hard (and very expensive) version.

"But it is certainly alive in that niche market, and getting some new design wins for space & defense use. We are in the process of adapting our SwiftX cross-compilers to support it" ...

"**ShBoom** survives as the PSC1000, and appears to be thriving thanks to being promoted as a 'Java chip'."

http://www.forth.com

A variant of **Chuck Moore's P32** has been designed expressly for packet switching. The P32 router was designed to route packets and provide protocol translation on multiple gigabit data streams on a chip that could be mass produced for around $1.

Jeff Fox has re-organised his **UltraTechnology web site** and added new material. Current work focusses on the F21d which is performing slightly better than expected.

http://www.UltraTechnology.com

fcebcab@ciccp.es

The 21st FORML Conference Forth and the Internet

Federico de Ceballos

I am grateful to Federico for this personal report of FORML, which I think is the first to appear in Forthwrite.

This last November, I took the opportunity of 'crossing the Pond' and visiting sunny California, the home of FIG. I went not only to present a paper at the 21st FORML Conference but also to have first-hand knowledge of the main characters behind the scenes. It did pay off: I met Chuck, Elizabeth, Ting and a number of other Forthers. What follows is a short remembrance of what happened there.

A Long Journey

For me, getting to the Conference has been the longest journey I have ever made in such a short time. To arrive at the site I had to go down to Madrid, up to London, west to Los Angeles and up again to Monterey.

Of course, this is not the easiest way, but it allowed me to be at the 20th Anniversary Reunion and also do some shopping at London. At the same time, flying to Los Angeles instead of San José gave me the opportunity to do some driving along the Pacific coast.

The Monterey Peninsula and Pacific Grove

The conference was held at the Asilomar Conference Center, a camp and conference site for the YWCA dating from 1913 and now operated by the State of California. The name itself is a coined term, trying to mean 'a refuge by the sea' in Spanish (well, not quite...).

The Centre is a complex of meeting halls, dining facilities and guestrooms situated on 105 secluded acres of scenic forest and sand dunes. It is a curious place, with buildings having names the like of 'Scripps', 'Crocker' or 'Manzanita' (this last is real Spanish, it means little apple) and a lot of little roads which can make you lose your way at night.

The XXI Conference

The first conference was held in London in January 1980, but moved the following year to Asilomar and has stayed there since.

Robert R. Reiling, who had directed the conference for the last nineteen years, died recently and was duly remembered at the beginning of the first session. Richard C. Wagner has taken the torch for this year, but it appears that there will be greater rotations in years to come.

Papers Presented

Wil Baden talked about one of his favourite topics: data encryption. In a paper called SHA-1 Secure Hash Algorithm, he presents an ANS-Forth implementation of that algorithm trying to get the simplest code from a reader's point of view. Even if you are not interested in this subject, it is a fine example of Wil's coding style.

In another paper, Wil talked about "the most powerful text editor that he's ever used". It is written in Forth, is directly integrated with Forth and has the ability to submit a line of text to be interpreted. This way he doesn't have to leave the editor, since almost all work can be done in it.

Yet he had something more to present, and this was Solitaire. This is not a card game, but a powerful way of encrypting messages. Having in mind the dictum "cheap, fast, good: choose two", he left aside the middle one and produced an algorithm, which doesn't need a computer. Of course, it does need a pack of cards. As Wil puts it: "soon the secret police will seize and torture anyone with a deck of cards. A card deck is a munition and criminal to export".

As an alternative, of course, you can use your Forth system.

Apart from all this, Wil and his anagrammatical friends Neil Bawd and Albin Dew presented some impromptu papers. The shortest one showed how to determine whether a machine is big-endian or little-endian.

Wil's (or Albin's) solution is: `: LITTLE-ENDIAN ( -- flag ) BASE C@ 0<> ;`

Glen B. Haydon discussed Forth Philosophy in the real world. He made a short walk through the history of the language and the conflict between natural change and standardisation.

Everett F. Carter explained how to subset TCP/IP. He did a short review of the different layers involved and showed what kind of applications could benefit from using one or more layers instead of the whole lot.

Of all papers, the one that had the least relationship with Forth was the one presented by **Philip J. Daunt**. He is a practising attorney whose only relationship with computers is as an end user. He talked about the problems that may be encounter by someone who tries to start a new business (or the ones that might appear later, if he or she chooses to go back to their previous job).

John R. Hart gave a rather technical presentation about building a network transceiver using Forth as an HDL. Thanks to Forth's versatility, he manages to get a clean solution that could be designed in three days and fits into a low cost PLD.

Charles Esson presented several papers about ColdForth, a Forth OS for the ColdFire family of processors. In the first one, he explains the design goals of ColdForth and the kernel details of his TCP/IP implementation.

Another paper of his describes ColdForth Multi-tasking. This system employs a pre-emptive version as opposed to the co-operative behaviour found in most systems. By using pre-emption he is able to assign a low priority to the TCP/IP stacks. The author goes on and discusses the problems involved in generating a time-based waveform in a heavily loaded system.

In his last paper, Charles describes ColdForth heap management, something that is necessary with the TCP/IP protocol. He is primary concerned with avoiding fragmentation and this is achieved by a simple way (even if not necessary the most memory efficient).

In ColdForth, memory buffers are memory blocks with a size that is a power of two. Furthermore, the address of a memory block has bits below its size set to zero. This way, it is easy to split a block into to equal ones or to join them again when both have been freed.

C. H. Ting presented several papers, one of them being quite short. It has a short title ('Tao of Forth'), a short summary (it states that this paper attempts to simplify Michael Ham's description of Forth[1]) and a text shorter still.

The text proper is reproduced to the right of this paragraph. I don't have Ting's written permission to reproduce the entire paper, but I believe that you cannot copyright a character. With this symbol, the author wants to convey the idea that the essence of Forth lies in its ability to define new words.

:

爲學日益
爲道日損
損之又損
以至于無爲

Fortunately, the author was more explicit when talking. His sense of minimisation comes from the Tao Teh King. Being Chinese himself, he was able to quote directly from Chapter XLVIII (reproduced to the left). The symbols mean something like 'do learn daily increase', 'do Tao daily decrease', 'decrease and decrease until nothing' and 'do no do and no no do'. I hope this makes more sense.

In another paper, Dr. Ting presents eForth v2, the first serious enhancement since 1990. The main changes concern speed improvements (achieved by means of using low level words for more utilities and switching in some processors to subroutine threading) and simplification of the most complex aspects (name and code space are now mixed, CATCH and THROW are not implemented, there is only one vocabulary and multi-tasking is forgotten).

Ting argues that multi-tasking is required in very few embedded applications. My opinion in this matter is that one of the advantages of Forth is its ability to offer effective multi-tasking at a very low cost. It also happens that the number of small

[1] "Forth is like Tao; it is a Way, and is realised when followed. Its fragility is its strength; its simplicity is its direction."

tasks in an embedded application can be greater than the few main tasks found in desktop programs.

Have you ever wondered what does the 'e' stands for? It was neither 'easy' nor 'educational'. The program was to be called PIGForth, so the 'e' is the tail of the pig. From now on, however, eForth means embedded Forth.

As an extension to the new work done in eForth, Ting has prepared the "Firmware Engineering Workshop". This is a short tutorial that tries to teach hardware engineers the art of building firmware for embedded systems. The author is quite strong in his views:

> Twenty years ago when I was first exposed to Forth, I was completely convinced that it was the ultimate software tool that allows us human beings to realise the full potential of computers. I still have not seen anything coming even close to its expressiveness and power. [...] We have the tool. We only need the will to use it.

Ting devoted another paper to the P8 Microprocessor. This in a modification of the P16 (discussed in Volume 22 from Offete Enterprises) with an 8-bit external data bus. He manages to get a working computer with only the XS40 board, a XC4005 FPGA (with 5000 gates) and a 32KB RAM chip.

An interesting point in the design is the 'no-cost UART', which uses the right shift instruction 2/ to send data serially out and to receive serial data into the T register.

Elizabeth D. Rather spoke about how OOP has been incorporated into Forth Inc. compiler, in a paper called SwiftForth Foundation Classes.

My own paper, about accessing an Ethernet network from a Siemens AS990 system, appeared to be the longest in print. In it I describe a system which has a narrow application field (safety related systems in nuclear power plants) and the coding technique needed to access some utilities which were planned to be used from C functions and which relied in an operating system that, in my case, isn't there.

I won't say more about it here, but any further questions about my paper (or request for more information from others) can be directed to my e-mail address.

Richard E. Haskell talked about the philosophy of WHYP, a subroutine-threaded implementation for the 6812. The target contains the code (obviously), but the heads reside in a PC host. By means of a C++ program, the PC sends commands that are executed in the target (something similar to SwiftX line of cross-compilers).

All this is documented in Richard's new book: *Design of Embedded Systems Using 68HC12/11 Microcontrollers*, available from Prentice-Hall. The only question that arises is: why did he have to use C++ in the host? We all know of a better choice for this kind of task, don't we?

John Carpenter talked about mixing Java and Forth. Because Java takes some time to come up, he has decided to have Java up all the time and let it call Forth when needed. In his paper he explains the requirements (he is using SwiftForth) to make a DLL and shows a simple example of its usage.

John Rible presented a short paper about CARDIAC 2000. CARDIAC is a cardboard illustrative aid to computation that was developed at Bell Labs in the late '60s to be used in various classroom settings. John has prepared a new instruction set based on the technology of the '90s.

Charles Moore didn't give a proper presentation. Instead, he talked about what he has been doing lately. This is mostly programming in Color Forth. He showed us one screen of code and complained about Color Forth not having a large number of followers.

This is a good topic for a flame war (and indeed, shortly after the conference, Wil Baden posted a message in comp.lang.forth stating that Color Forth was "three years waste of Chuck's time"). Whatever your view may be, hearing its inventor talk about it you get the feeling of "something" being there.

He also gave an impromptu talk about **100x**. This was a reflection about the relationship between secondary storage space and main memory, which has kept more or less constant through time (640 KB memory and 40 MB disk, or 64 MB memory and 6 GB disk).

One thing I noticed about Chuck: During the first session, he was sitting by my side (actually, I was the one who sat by his side). The next session, however, he moved to another place far away. The following day he moved yet again and he did the same in the rest of the sessions. The last day he remarked that everybody else appeared to have sat in the same place throughout the conference. He thought that sitting in a different place gave a different perspective. This may sound funny, of course, but it gives an idea of how far you should go when you are interested in getting the "whole picture".

Life After Forth

Contrary to what this heading says, there doesn't appear to be much life after Forth (at least in FORML). We spent all the available time talking about Forth-related subjects; not only during the 'Wine and Cheese' parties but also afterwards until two or three in the morning.

Anything was allowed, from our opinions towards Microsoft (ah, ah) or when should the next millennium be celebrated.

Other Issues

It appears that FIG is having some trouble with keeping their current members and getting new ones. Their grand total is now around seven hundred and they think that they might have a problem if that number is allowed to decrease further. One of FIG's assets is Forth Dimensions, which could be compromised if it doesn't have adequate funding.

Another problem is FORML itself. This conference managed to attract quite a few papers and two dozen attendees. However, as you can see in the previous pages, several were from the same author. It appears that some people are more interested in renewing friendships that in "Forth Modification".

Everybody is encouraged to attend this year, if only for the fun. Of course, it is a long journey from here...

It is a custom at the conferences to award some prizes for various reasons. This year, bottles of Chilean wine were given to the shortest paper (Ting, of course), to punctuality (it went to Elizabeth Rather, who managed to arrived in time directly from France after some delays at the airport) and to the person coming from furthest away (this was given, ex aequo, to Charles Esson - from Australia - and myself).

As an additional memento, newcomers were given the FIG T-shirt. I'm looking forward to the summer; but then I suppose we all are.

Chris Jakeman
cjakeman@bigfoot.com

Did you Know? - Novell

While other parts of Forthwrite bring you all the news and the latest ideas and developments, the **Did You Know?** section highlights achievements in Forth, both recent and historical (taking care always to distinguish hearsay from attested fact).

Initially I found this item difficult to credit but it does check out.

The original Novell network system was written in Forth (1980-81 or so) on special S-100 computers - Dean Miller

Dean reports: " When I booted the Novell network it came up with the Forth prompt and I was able to execute normal Forth words.

Note that this pre-dated the PC. It was S-100 (Z80) hardware with dumb terminals. There was a server box and client boxes. I don't recall what network hardware or physical layer protocol was used.

I also don't recall whether or not the client computers were running Forth as the OS or were running CP/M. I wrote a couple of add-ons for it (the network was being installed in an attorney's office in the Denver area)."

Jeremy Fowell
0121-440-1809
Jeremy.Fowell@btinternet.com

F11-UK, FIG UK Hardware Project

Jeremy Fowell

A brief update as Jeremy reaches the end of his "to do" list.

Documentation Progress

I now have the Glossary for Pygmy HC11 finished in ASCII text format. It seemed to take forever. It contains around 300 entries spanning 19 A4 pages. Fortunately I now have Adobe Acrobat on my PC and so intend to provide all documentation in PDF format. It should bring that 19 page total down a bit I hope.

Not all of these items are executable Forth words, I have tried to include system constants and anything which might crop up in the code, even though there are a number that you would never use in any application programs that you write.

Stepping up onto soapbox ... There is nothing worse than coming across an obscure name in some Forth code that takes ages to track down and explain. I'm sure this sort of thing causes quite a few newcomers to give up on Forth altogether; it's certainly caused me a few problems.

On-Board A/D

The code for the on-board 8-bit A/D converter is up and running.

At first I noticed quite a bit of quantisation noise (is that the correct term ?) when I connected a potentiometer to an input and displayed the result on the PC screen. The value kept hopping around by more than +/- 1 digit and wouldn't settle. After various attempts at smoothing the value I hit upon the rather obvious idea of using the built in function where 4 consecutive readings are taken and saved in separate memory locations. These are then placed on the stack with `A/D@` and their average calculated by:

```
: 4FILTER ( b1 b2 b3 b4 -- b5 )
+ + + 4/  ;
```

`4/` is a code word I have written to avoid normal division (which is a bit slow) by using the assembler shift-right instruction.

The complete phrase is:

```
VOLTAGE-INPUT READ-ONCE A/D@
4FILTER
```

Text Output From LCD

There is also code now to drive a 16 character x 2 line LCD. The unit I am using has a LED backlight (80 mA - hardly low power), and looks very smart indeed. It is connected to the HC11-E1 via the SPI serial bus using a single 74HCT595 shift register. The maximum SPI clock rate of 1 MHz is used and it worked first time.

Well, almost. In fact I spent more than one evening struggling to get it to display anything at all. It turned out that the DWOM bit (where do they get these names from ?) in the SPI control register had been overlooked. It has to be cleared to zero to select push-pull outputs on Port-D rather than open-drain (slow rise time with 10 k pull-up resistors).

The code uses a special version of `EMIT`, `LCD-EMIT` which sends characters to the LCD rather than the PC screen. To make things as simple as possible I now have to write 2 code words, `>LCD` and `>SCR`, which will quickly redirect output to the LCD or back to the PC screen. Then most of the existing output words such as `TYPE`, `.` , `U.` and `U.R` etc. can still be used. The source code is included in a file of extensions as part of the F11-UK kit.

The LCD also demonstrates the principle of adding another 8-bit output port to the F11-UK board using the SPI bus.

I did the LCD work at this point because I needed it for a commercial project I am also working on.

However what we **really** need is for the documentation to be finished off ASAP . . .

Help Wanted

Graeme and anyone else who might know, I have tried to post a message to the Motorola HC11 News Group using the email address:

68HC11@oakhill-cisc.sps.mot.com

and it bounced straight back with: Returned mail: Host unknown (Name server: oakhill-cisc.sps.mot.com: host not found).

Could you help please ?

Fred Behringer
behringe@mathematik.tu-muenchen.de

32-bit GCD without Division in ZF and Turbo Forth

Fred Behringer

Fred's work is familiar to all of us and here he shares with us his twin passions of Forth and mathematics, endeavouring as always to make them accessible to the largest possible audience.

Forth For more Fun

There is much ado about tutorials for the novice. Yet, what is this novice like? There are so many "novices" out there who aren't really novices. Novices they are only with respect to Forth. The best "tutorial", so I hold, is a small program which the novice can test and analyse.

This is a "fun program" in the sense of Hugh Aguilar (see Forthwrite Issue 105). It surely has a definite purpose but it was fun writing it and nobody told me to do so.

The program below is a Forth version of an algorithm in the book "Prime Numbers and Computer Methods for Factorization" by Hans Riesel (Boston, Stuttgart, 1985). It's for determining the greatest common divisor (G.C.D.) of two 32-bit integers. The special feature of this program is

"it doesn't need division"

that is doesn't need division. All that's needed is done by subtractions and binary shifts only. (Divisions need 40 CPU cycles whereas subtractions can do with 1 CPU cycle. With Pentiums, the ratio still is 10/1.)

I'll take Forth (ZF and Turbo Forth) for the purpose of comparison only. My main purpose is to show how directly machine code can be incorporated in a Forth program, and 32-bit machine code in a 16-bit Forth at that. I don't care about portability. I'm working with a PC clone. The G.C.D. is a basic mathematical concept sophisticated enough so that casting it in a CODE definition is vital. I'm not afraid of reinventing the wheel. If anybody can tell me where exactly this approach has been published in the Forth literature, I would not hesitate to renounce priority.

My second purpose is to show how easily 32-bit code can be incorporated in a 16-bit Forth. The whole package of "32-bit extensions" that I needed with respect to the (16-bit) ZF or Turbo Forth assembler was done in form of two (Yes only two!) extra colon definitions.

I'm fully aware of the fact that what I'm doing is just using (mis-using?) Forth as an easy and easy-to-modify way of assembling and using machine code.

There are many papers on code optimisation in the existing Forth literature. The best optimisation, however, is the one that optimises the algorithm underlying the code. So let's take this as a challenge to everyone whether he or she can find the G.C.D. in less time or by using less code.

The Algorithm

The program first tests whether the entries are positive. Any negative entry is converted to its positive counterpart. If at least one entry is zero, the program skips to the end and yields zero as the "greatest common divisor".

With respect to the algorithm, I take the liberty of quoting from the book just mentioned:

"Another way of avoiding division and multiplication of multi-precision integers is to apply the binary form of Euclid's algorithm. It uses subtractions only, coupled with the fact that it is particularly easy in a binary computer to divide integers by a power of 2. The integer has only to be shifted `s` positions to the right to effect division by 2^s. The scheme for finding `d = (a,b)` according to this algorithm is:

1. Suppose that `a` ends in `u` binary zeros and `b` in `v` binary zeros.
2. Let `d` = $2^{\min(u,v)}$.
3. Form `a' = a/2`^{u} and `b' = b/2`^{v}, i.e. shift `a` and `b` to the right until all binary zeros at the right ends are shifted away.
4. Now both `a'` and `b'` are odd numbers. Form `c = a'-b'` which is even. Shift away the trailing zeros of `c` and repeat step 3, this time with the reduced `c` and `min(a',b')`. When the result of the subtraction becomes 0, the current value of `a'` is the largest odd factor `d'` of `(a,b)`.
5. Finally, put `(a,b)` = `dd'`.

It is quite easy to show that the binary form of Euclid's algorithm takes at most about `ln max(a,b) = 3.32 * log max(a,b)` steps.

However, as in the case of the ordinary Euclidean algorithm, the average number of steps is smaller, approximately `2.35 * log max(a,b)` and thus, because of the simplicity of the operations performed in each step, the binary form competes very favourably with the standard version of the algorithm." - End of quotation.

The program was tested with ZF and Turbo Forth. There was no noticeable difference between the two. The program was also tested with two different machines, an 80486/66 and a K6-2/350. The ratio of execution times exactly reflected the ratio of CPU operation frequencies.

I have tested the worst case behaviour only. To achieve this, I've taken 31 one bits for the first entry (2,147,483,647 decimal) and 30 one bits for the second (119,304,647 decimal). These two entries are relatively prime, i.e. their G.C.D. equals 1, and the determination of the G.C.D. takes the maximum number of steps, i.e. 31 . Furthermore, I have employed

```
: XXX 30000 0 DO GCD LOOP ;
: YYY   100 0 DO XXX LOOP ;
```

for determining the overall time of executing 300,000 times the GCD , and

```
: XXX 30000 0 DO LOOP;
: YYY   100 0 DO LOOP;
```

for subtracting the loop overhead in order to get to the actual execution time of GCD.

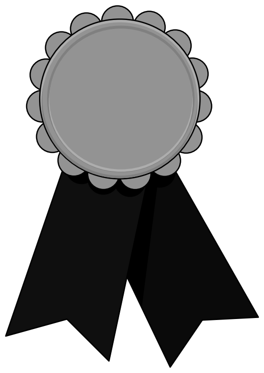
The numbers are given in decimal. In order to avoid stack overflow and stack overhead, I have added two immediate stack entry `PUSH` operations to the program's entry `POP` operations, and deleted the `PUSH`ing of the G.C.D. result on the stack. The execution times of `PUSH` and `POP` can be neglected. Here are the worst case test results:

Execution time of GCD is
- 157 microseconds on the 80486/66 machine, and
- 21.5 microseconds on the K6-2/350.

Following is the program listing (works in ZF as well as Turbo Forth):

```
\ Greatest common divisor (G.C.D.).
\ Stack entries = 32 bit signed integers.
\ Negative integers will first be converted to positive ones.
\ Zero integers will lead to a zero G.C.D. result on the stack.

ONLY FORTH ALSO
ASSEMBLER DEFINITIONS
HEX

\ Only two assembler extensions needed. Anything else with respect
\ to the 32-bit register access can be done with the ZF (or Turbo
\ Forth) inherent 16-bit assembler

: OPSIZE: 66 C, ;          \ 32 bit data access;
                           \ EAX instead of AX, etc.
: BSF ( xX yX -- )         \ bit search forward;
                           \ yX shows first 1 bit in
     OF C, MOV             \ xX (from the right)
     OBC HERE 2- C! ;      \ replace MOV opcode by 2nd part of BSF opcode

FORTH DEFINITIONS

CODE GCD ( ud1 ud2 -- ud3 )
     10 # CL MOV           \ load CL for 2 byte shift
     OPSIZE: BX POP        \ ud2 in EBX
     OPSIZE: BX CL ROL     \ swap for "little endian"
     OPSIZE: AX POP        \ ud1 in EAX
     OPSIZE: AX CL ROL     \ swap for "little endian"
     OPSIZE: DX DX XOR     \ EDX = 0
     OPSIZE: DX INC        \ EDX = 1
             HERE          \ jump address for JS
     OPSIZE: AX NEG        \ renders AX positive
             JS            \ repeat if negative
             HERE  5 +
             JNE           \ if EAX = 0, then
     OPSIZE: DX DX XOR     \ EDX = 0
             HERE          \ jump address for JS
     OPSIZE: BX NEG        \ renders BX positive
             JS            \ repeat if negative
             HERE  5 +
             JNE           \ if EBX = 0, then
     OPSIZE: DX DX XOR     \ EDX = 0
     OPSIZE: DX BX XCHG
     OPSIZE: BX BX OR      \ if at least one input = 0,
     0<>                   \ then jump to the end,
        \ ---------------------------------------------
     IF                    \ with the result DX = 0
```

```
  OPSIZE: BX DX XCHG
  OPSIZE: BX CX BSF          \ e.g.: BX=8 -> CX=3
          CL DL MOV          \ save CL
  OPSIZE: BX CL SHR          \ BX = BX/2^CL
  OPSIZE: AX CX BSF          \ e.g.: AX=4 -> CX=2
  OPSIZE: AX CL SHR          \ AX = AX/2^CL
          DL CL CMP          \ CL >= DL ?
          HERE  4 +
          JGE                \ if not, then CL -> DL
          CL DL MOV          \ DL = min(CL,DL) in any case
   \ ------------------------------------------------
  BEGIN
    OPSIZE: AX CX BSF        \ e.g.: AX=2 -> CX=1
    OPSIZE: AX CL SHR        \ AX = AX/2^CL
    OPSIZE: BX AX CMP        \ AX > BX ?
            HERE  4 +
            JG               \ swap if not
    OPSIZE: AX BX XCHG       \ now AX >= BX at any rate
    OPSIZE: BX AX SUB        \ AX = AX-BX
    0=
  UNTIL
  \ ------------------------------------------------
          DL CL MOV
  OPSIZE: BX CL SHL          \ BX = BX*2^DL
  10 # CL MOV                \ load CL for 2 byte shift
  OPSIZE: BX CL ROL          \ swap for "little endian"
THEN
\ ------------------------------------------------
OPSIZE: BX PUSH              \ BX = ud3 = G.C.D.
NEXT END-CODE
```



Nominations for the FIG UK Awards

The FIG UK Awards of 1998 were won by Philip Preston and Paul Bennett. These awards are given to encourage effort and recognise achievement. Now is the time to look back and send in your nominations for 1999.

Free membership

To nominate your candidate, send in a note of who, in your opinion, most deserves an award and why. The recipient of each award will receive a place in FIG UK web-site's Hall Of Fame, a mention in Forthwrite and ***a year's free membership***.

Achievement

The Achievement Award is given to the member who has made the best contribution towards Forth during 1999. The contribution may include a published article, a library of code or an idea which inspires others. Whatever form it takes, the contribution must support the goals of FIG UK.

Forthwrite

The Forthwrite Award is given to the member who has made the best contribution to Forthwrite during 1999. The contribution may be judged on quality of writing, tutorial potential, entertainment value or any criteria which seems appropriate.

The awards are judged by the officers of FIG UK. All who are members on 31st Dec. 1999 are eligible candidates (except the judges).

Jack Brien
jackbrien@bmallard.swinternet.co.uk

All you need to know about STATE, IMMEDIATE and POSTPONE

Jack Brien

The issues which Jack reviews here by special invitation have been the source of confusion and many postings to comp.lang.forth. Jack provides us with the definitive reference to this thorny topic.

Using and Controlling STATE

STATE is the flag which tells programs whether the text interpreter is interpreting or compiling (i.e. making a new definition).

```
STATE @ IF compiling ELSE interpreting THEN
```

While compiling, STATE can be temporarily turned off with **[** to begin interpreting, and restored with **]** to return to compiling. This may be done, for example, to calculate a literal value, e.g.

```
VARIABLE FOO
...
: BAR ... [ foo @ ] LITERAL ;
: BAR1 ... foo @ ;
```

"Foo @" is interpreted at *compile time* in the definition of BAR, and the result compiled by **LITERAL**. In the definition of BAR1, "foo @" is compiled to be executed later, at *run time*. BAR always returns the same initial value of FOO, whereas BAR1 returns whatever FOO holds when it is executed.

' *name*	returns *name*'s *execution token* (xt) which represents its behaviour
' *name* **EXECUTE**	performs *name*'s behaviour

EXECUTE does not either know or care about the current STATE, so, given the same xt, it will perform the same behaviour whether interpreting or compiling. That's how the text interpreter treats **IMMEDIATE** words, and normal words when interpreting. If it encounters a normal word while compiling, the text interpreter adds its behaviour to the current definition. The Standard word to do this, given an xt, is **COMPILE,** .

Immediate Words

When an immediate word is met during compilation, it is executed instead of compiled. They are easy to create - all you have to do is use the word **`IMMEDIATE`** directly after the definition. (`IMMEDIATE` changes the most recent definition). You can have `IMMEDIATE` colon definitions, `IMMEDIATE` variables, `IMMEDIATE` constants, `IMMEDIATE` anything that has a name. **`FIND`** can distinguish between immediate and normal words. It returns the xt and a flag of -1 for normal words; the xt and a flag of 1 for immediate words. That is the only difference between normal and immediate words. Neither `EXECUTE` nor `COMPILE,` know or care if the xt they receive comes from a normal or an immediate word.

Here's a simple text interpreter (so simple, it gives up at the first unknown word):

```
BEGIN
BL WORD FIND DUP WHILE
   STATE @  IF                \ compiling?
      -1 = IF                 \ normal word?
      COMPILE, ELSE           \ add its behaviour to the definition
      EXECUTE THEN            \ immediate - perform its behaviour
   ELSE                       \ interpreting
   EXECUTE THEN               \ perform its behaviour
REPEAT
2DROP
```

That's all you need to create a simple compiler on most Forths. In fact, you would also need just one immediate word - **`[`** - to switch from compiling to interpreting. Normal words can switch in the other direction. But it would be more convenient to have a number of immediate words:

▪ Structure Words:	**`BEGIN`** , **`DO`** , **`IF`** , **`THEN`** , etc.
▪ Literal Words:	**`[']`** , **`[CHAR]`** , **`LITERAL`** , **`S"`** etc.
▪ Words to stop compiling:	**`[`** , **`;`**
▪ Comment Words:	**`(`** , **`\`** , **`[IF]`** etc.

Compile-Only Words

However, you will notice that most of these words have behaviours that make no sense while interpreting. Only the comment words can actually be executed regardless of `STATE`. The rest are "compile-only" words which should never be invoked (by executing them either directly, or in a definition in which they've been compiled) when interpreting. It's possible to implement compile-only words as normal or immediate definitions (and most Forth systems do) but you have to be careful never to invoke them in the state for which they were not intended.

Rule 1: Never try to get or use the xt of a compile-only word

It's not good practice - and on some systems it may not even be possible - to re-define compile-only words. Fortunately, these restrictions only matter if you want to write your own compiling text interpreter - and that's outside the scope of the present article.

Using POSTPONE

There is a Standard way to get a compile-only word's compilation action:

```
POSTPONE name
```

POSTPONE itself is also compile-only.

If *name* is compile-only, its compilation action is added to the current definition.

If *name* is immediate, code equivalent to **' *name* COMPILE,** will be executed at compile-time.

If *name* is a normal word, will be code equivalent to **' *name* COMPILE,** will be executed at run-time.

In any case, code compiled by POSTPONE should itself never be invoked while interpreting. The only possible exception is where *name* is an immediate word:

```
: MY(     POSTPONE  (  ; IMMEDIATE
```

will work on every system I know, because it could be equally well defined as:

```
: MY(     ['] (  EXECUTE ; IMMEDIATE
```

which is better practice.

So, does the use of POSTPONE make a definition compile-only? Not quite. A simple example comes from the Standard document:

```
: NOP  : POSTPONE ;  IMMEDIATE ;

NOP ALIGN    NOP ALIGNED
```

NOP is a defining word that creates do-nothing colon definitions and makes them immediate, so they do nothing regardless of STATE (they are never compiled into other words). Here STATE gets switched when NOP executes - **:** turns it on, and the POSTPONEd **;** turns it off . So although NOP performs the compilation action of **;** it does so when STATE is ON.

Rule 2: Words defined with POSTPONE are usually compile-only.

The same principle is used in so-called STATE-smart words.

State-Smart Words

STATE-smart words are a high-level convenience. They are good when you don't want the user to have to remember two separate words - one for interpreting and one for compiling. So F83 had a word ASCII which could be defined:

```
: ASCII   STATE @ IF   POSTPONE [CHAR]   ELSE  CHAR  THEN ;
IMMEDIATE
```

The compilation action of ASCII is guarded by the test of STATE. When compiling, it's equivalent to [CHAR]. When interpreting, ASCII invokes CHAR instead. It's also safe to POSTPONE ASCII and other STATE-smart words, so long as you observe the restriction on never invoking the resulting code while interpreting. Then what you have POSTPONEd will always use the compilation action, since STATE will always be on when it's invoked.

Otherwise, here's an example of what can go wrong:

Say you want a word CONTROL that returns a control code instead of an ASCII value.

Interpret version:

```
: CONTROL   POSTPONE ASCII     \ OK. ASCII is an immediate word
            >CONTROL ;         \ Convert the ASCII value returned
```

Test it and it works. CONTROL invokes CHAR while interpreting, and it also compiles into other definitions which also work while interpreting. However because ASCII is state-smart, the phrase POSTPONE ASCII can lead to problems later on. For example, we can build a compile-only version [CONTROL] - by analogy with

```
: [CHAR]   CHAR POSTPONE LITERAL ; IMMEDIATE
```

we get:

```
      : [CONTROL]   CONTROL  POSTPONE LITERAL ; IMMEDIATE
```

Doesn't work as you might expect. CONTROL only invokes CHAR **if STATE is OFF**. When it is executed by [CONTROL] it will invoke the compilation action of [CHAR], which is certainly not what is required.

It wouldn't have helped to write

```
      :  CONTROL  ['] ASCII EXECUTE ;
```

['] , **'** or **FIND** cannot unaided distinguish a STATE-smart word from any other immediate word, and so extract the STATE-dumb interpretation action. There's no easy way to invoke the interpretation action of a STATE-smart while compiling. **The safest option** is to treat STATE-smart words like compile-only words and not mess with their xt's.

Rule 3: State-smart words defined with POSTPONE are always compile-only.

COMPINT below is an un-easy option, inspired[1] by Anton Ertl's combined words at http://www.complang.tuwien.ac.at/forth/combined.zip It's messy, but it's a good illustration of the power of POSTPONE. COMPINT is an enhanced POSTPONE which will avoid these problems:

```
      : [[  POSTPONE  [  ;      \ Turn STATE off at run time, not compile time

      : COMPINT                 \ Compile code equivalent to the interpret
                                \ action of a STATE-smart word
      '                         \ Get the xt
      POSTPONE STATE  POSTPONE @ POSTPONE IF
         POSTPONE [[            \ Code to turn STATE off if it is ON
         DUP COMPILE,           \ Compile the xt
         POSTPONE ]             \ and restore STATE
         POSTPONE ELSE          \ Interpreting already; don't mess with STATE
```

[1] See also Anton Ertl's comments http://www.complang.tuwien.ac.at/forth/dpans-html/comment-semantics.html. and Mitch Bradley's proposal on 'syntactic elements' ftp://ftp.minerva.com/pub/x3j14/proposal/99-032-proposal-impl-q5-7-8-9.txt for a different perspective.

```
      COMPILE,                  \ Just do it
   POSTPONE THEN ;  IMMEDIATE
```

Used as **`: CONTROL  COMPINT ASCII  >CONTROL ;`** this produces the code:

```
   : CONTROL  STATE @ IF [[  ASCII ] ELSE ASCII THEN >CONTROL ;
```

Rule 4: `COMPINT` is a safe version of `POSTPONE`.

Better still, avoid state-smartness altogether and just factor the interpretation action into a definition of its own and compile that instead!

This brings us to the last and simplest rule:

Rule 5: Avoid state-smart words.

01932 786440
101745.3615@compuserve.com

Vierte Dimension 1/00

Alan Wenham

Alan provides a look at the latest issue of the German FIG magazine. To borrow a copy or to arrange for a translation of an individual article, please call Alan.

Miscellaneous

Henry Vinerts

Letters, publicity inserts for FIG UK, FIG USA and Dutch FIG together with a report from Henry Vinerts on recent meetings of Silicon Valley FIG.

Structural Resonance Analysis

Johannes Reilhofer

Reprint of a report by Johannes Reilhofer from the last German Forth meeting in Oberammergau concerning the non-destructive vibration testing of automobile drive shafts. The test system was programmed in Forth and has found approval among several American firms.

CF2NAME

Wolfgang Allinger

This inspiring article is by Wolfgang Allinger, well known to the members of Forth Gesellschaft. It was designed for F-PC but is also fundamentally applicable to other Forths and it concerns multi-tasking. When one is in the interactive testing phase of a program which is not yet working, one often has great need to obtain the name of the calling word (the word which may be causing the system to hang) from the addresses left on the return stack. It is a type of “inverse problem”.

Hashing

Hugh Aguilar

This is the second and final part of an article by Hugh Aguilar, translated by Fred Behringer, which originally appeared in Forth Dimensions, Vol 20, Nos 5 & 6, of Jan/Apr 1999. It covers the interactive breaking of coded text with keys of the type used in the American Civil War. These are rich Forth programs.

Forth For Fun

This is the German version of the item which appeared in Forthwrite 105.

Reed-Solomon Error Correction - Part 2

Glenn Dixon

This is the second part of the paper by Glenn Dixon, translated by Fred Behringer, which appeared in Forth Dimensions Vol 20, Nos 5&6, of Jan/Apr 1999. It concerns error recognition and correction in ZIP drive operations, compact discs etc..

From The 'Net

This snippet came from comp.lang.forth. I love this sort of thing.

```
> The ability to set BASE to anything from 2 to 36 is useful for
> giving a tutorial on number bases but I wonder if it has any other
> utilitarian value.  In practice do any of you people use number
> bases other than 2, 8, 10 and 16?  If so can you tell us about it?

How about converting a time in seconds to a string HH:MM.SS? As in:

      DECIMAL

      : SIX  6 BASE ! ;

      : .HMS ( n -- )
         0 <# DECIMAL # SIX # [CHAR] . HOLD
              DECIMAL # SIX # [CHAR] : HOLD
              DECIMAL # # #> TYPE SPACE ;

      : >SEC  ( h m s -- n )
         ROT 60 * ROT + 60 * + ;

      10 30 5 >SEC .HMS   ( 10:30.05 )
```

John Tasgal
0161 7739365
john@tcl.prestel.co.uk

An Introduction to Machine Forth

John Tasgal

There has been a lot of interest in Chuck Moore's recent work on special processors and the languages that go with them (see issue June 1999 for an interview with Moore). These are Forths that deviate from the classical model to match the hardware more closely. The differences challenge our assumptions about standard Forth; could it become both simpler and better?

John Tasgal has researched both Machine Forth and Color Forth and expounds these differences. Alongside the Color Forth article (in the next issue) is a commentary on some of Chuck's published code, showing how complex code can be written with a simpler Forth.

Machine Forth (MF) is a development, principally by Charles Moore and Jeff Fox, of classical Forth. Its aim is to simplify both the design of stack chips and the Forth-style languages they use. It is a low-level Forth closely mapped to the underlying hardware.

There are two quite separate parts to this language:

The first part is a core which is the instruction set of a MISC (Minimal Instruction Set Computing) chip. The second part is an extension to the instruction set to allow word and dictionary building etc.

As far as I know there is no standard and so I have chosen to use the Ultra Technology F21 chip as the 'reference' for the MF instruction set. This article describes the 'programming model' of a MISC chip. I only describe hardware where relevant.

Notation

T(n)	The n'th bit of register T
T(n1 .. n0)	A bitfield in register T from bit n1 down to bit n0

Phrases

To present the structural templates below in compact form these abbreviations are used for phrases, that is, a sequence of tokens :

`flag?`	A phrase which leaves a value in T(19..0) for use by `IF`
`carry?`	A phrase which leaves a value in T(20) for use by `-IF`

`<tt`	The phrase executed when `IF` is true
`<ff`	The phrase executed when `IF` is false

The MISC Chip

- There are 5 registers, 2 circular stacks, and a 5-bit opcode with 27 instructions decoded.
- All on-chip registers are 21 bits wide.
- The MSB, bit 20, is used for memory control for instructions which access external memory; as carry for the add instruction; and as an ordinary bit for the others.
- `IF` reads bits 19..0 and jumps if they are false. It does not pop the stack (unlike a classical `IF`). It is therefore called 'non-destructive'.
- `-IF` jumps if bit 20 is false. This too is non-destructive.
- `2/` on the F21 shifts all bits T(20..0) right. It can therefore be used either as `2/` or `U2/`, or for multiple-precision arithmetic.

The Registers

- **PC** The Program counter
- **A** The Address register for memory access
- **T** Top of data stack, the implied operand for arithmetic, logic and `IF` instructions
- **S** The 'subtop' register, the second on the data stack.
- **R** Top of return stack

The Circular Stacks

- **The Data Stack** **(S2 .. S11)** A 16-element circular stack below T and S
- **The Return Stack** **(R1 .. R10)** A 16-element circular stack below R

The Instruction Set

Control

- **`ELSE`** Unconditional jump
- **`IF`** Non-Destructive `IF`. Jump if T(19..0) is false (leaves stack untouched)
- **`-IF`** Non-destructive jump if-carry-false

- **CALL** A Subroutine call. Push PC+1 to R
- **RET** Return from Subroutine. Pop R to PC

A Register

A	**(-- A ; T = A)**	Push A to T
@A	**(-- n0 ; T = ^A)**	Fetch contents of memory at address A and push to T.
@A+	**(-- n0 ; T = ^A, A=A+1)**	Fetch A and push to T. Increment A. ('Auto Post-Increment')
!A	**(n0 -- ; mem(A) = n0)**	Pop T to memory at address A
!A+	**(n0 -- ; mem(A) = n0, A=A+1)**	Pop T to memory at address A. Increment A
A!	**(a0 -- ; A = T)**	Pop T to A

R Register and the Return Stack

POP	(-- r0 ; r0 -R- ; T = R)	Pop R and push to T
PUSH	(n0 -- ; -R- n0 ; R = T)	Pop T and push to R
@R+	(-- n0 ; T = ^R, R=R+1)	Fetch from address in R, push to T. Increment R
!R+	(n0 -- ; mem(R) = n0, R=R+1)	Pop T to memory at address R. Increment R

Data Stack Manipulation

DUP	(n0 -- n0 n0)	Push T to T
DROP	(n0 --)	Pop T
OVER	(n1 n0 -- n1 n0 n1)	Push S to T

Arithmetic

+	(n1 n0 -- n0' ; T = T + S)	Add S to T.
+*	(n1 n0 -- n1 n0' ; T = T + S {T(0)=1})	If T(0) is true, add S to T non-destructively. A multiply step.
	(n1 n0 -- n1 n0 ; {T(0)=0})	If T(0) is false, do nothing.

Bitwise

▪ COM	(n0 -- n0' ; T = NOT(T))	Complement T. Invert each bit.
▪ AND	(n1 n0 -- n0' ; T = S AND T)	AND S to T
▪ -OR	(n1 n0 -- n0' ; T = S XOR T)	Exclusive OR S to T
▪ 2*	(n0 -- n0' ; T = T * 2)	Shift Left one bit. Write 0 to T(0)
▪ 2/	(n0 -- n0' ; T = T div 2)	Shift Right one bit. WriteT(20..1) to T(19..0). Write 0 to T(20).

Miscellaneous

▪ #	(-- n0 , | <number)	Fetch a number from PC+1 and push to T. Increment PC .
▪ NOP	( )	Do nothing for 1 cycle.

The Extensions

Very few words need to be added to an assembler based on the above instruction set to produce a working Forth system. The main categories are:

Definitions

▪ `:`	Colon starts a new definition
▪ `;`	Return. Does *not* end a definition
▪ `CREATE ... DOES`	To allow new types
▪ `CODE ... ENDCODE`	For machine code

Control Structures

These structures have the same meanings as Classical Forth but the flag/carry remain on the stack after execution.

▪ `flag? IF <tt THEN <ff`	If flag? is true execute <tt
▪ `carry? -IF <tt THEN <ff`	If carry? is true execute <tt
▪ `flag? IF <tt ELSE <ff THEN`	If flag? is true execute <tt, else execute <ff
▪ `carry? -IF <tt ELSE <ff THEN`	If carry? is true execute <tt, else execute <ff

▪ **`( index ) BEGIN ... NEXT`**	A loop with an single index
▪ **`BEGIN  flag?  WHILE <tt REPEAT`**	While flag? is true execute <tt
▪ **`BEGIN carry? -WHILE <tt REPEAT`**	While carry is true execute <tt
▪ **`BEGIN ...  flag?  UNTIL`**	Loop until flag? is true
▪ **`BEGIN ... carry? -UNTIL`**	Loop until carry? is true

These allow words of various types to be defined; a dictionary to be built; and for the control of program flow. Numerous other words for arithmetic, logic, and Operating System functions can then be added to this extension.

Differences From Classical Forth

The Semicolon

This doesn't end a definition; it means simply 'return'. Definitions run into one another.

The Address Registers

This moves addressing from the the data stack to a register, either A or R.
Both registers also have auto-post-incrementing instructions.
This changes the style of Forth as pointer arithmetic becomes the method of choice over the use of `DO ... LOOP`'s with indexes.

Non-Destructive Conditionals

In Classical Forth, `IF` destroys the top of stack. However in Machine Forth `IF`, and therefore all the conditionals based on it, are non-destructive. This removes the need to use `DUP` when conditionals repeatedly test a flag.

But, it may lead to more use of `DROP` to remove a flag which would have been destroyed by a conventional conditional. This suggests that the behaviour of other words and if necessary the program structure itself should be adapted to optimise the use of non-destructive conditionals, rather than simply copying a program written using the destructive versions.

This is another example of a change in programming style.

Tail-Recursion Optimisation

In any definition the return action of the word before a semicolon, and of the semicolon itself, can always be compiled into a single return.

```
    word1 ..... lastword ;
```

As nothing happens between **`lastword`** returning and '**`;`**' returning, the **`lastword`** return is superfluous.

A more elaborate example is the recursive call at the end of a `WHILE` loop. If we have a series of nested calls then the last instruction is in each case a return. At runtime this produces '**`; ; ; ; ;`**' viz. a sequence of returns.

The point is that when these calls unwind all that happens is that a sequence of returns are executed, one after the other. Nothing is done between them. The only necessary return is the first one pushed onto the return stack (and so the last to be executed).

Removing these superfluous returns is known as *tail-recursion optimisation*. Most Machine Forth compilers (and also Color Forth) contain a 'tail-recursion optimiser'.

Two syntaxes are currently in use to indicate that this optimisation is to be carried out:

- A special token '`-;`' (hyphen semicolon)
- A *smart semicolon,* which involves recognising the '**`lastword ;`**' pattern.

Tail-recursion optimisation is achieved through a compiler optimisation, and also by the syntax itself. The syntax is so designed that the programmer, simply by writing a semicolon after a recursive jump, causes the compiler to build a single return instead of nested returns. Therefore nested returns are eliminated *at the design stage* through a syntactical feature.

This is really a very unusual and elegant approach to this problem.

Next Issue

The remaining two articles in this series appear in the next issue of Forthwrite, describing Charles Moore's newest Forth, Color Forth, which builds on Machine Forth. This is followed by a detailed commentary on some of Chuck's Color Forth code to see how it is used in practice.

Dutch Forth Users Group

Reading Dutch is easier than you might think. And as Forth is an international language, reading Dutch code is easier still for a Forth enthusiast. Are you interested? Why not subscribe to

HCC-Forth-gebruikersgroep

For only 20 guilders a year (£6.30), we will send you 5 to 6 copies of our "fig-leaf" broadsheet 'Het Vijgeblaadje' . This includes all our activities, progress reports on software and hardware projects and news of our in-house products.

To join, contact our Chairman:
Willem Ouwerkerk
Boulevard Heuvelink 126
6828 KW Arnhem, The Netherlands
E-Mail: w.ouwerkerk@kader.hobby.nl

The easiest way to pay is to post a 20 Guilder note direct to Willem.

Chris Jakeman
01733 753489
cjakeman@bigfoot.com

From the 'Net - Cube Roots

Chris Jakeman

The extracts from comp.lang.forth repeated here show the newsgroup operating at its co-operative best. An unusual but real problem is posed, radically different solutions are offered from around the world and compared in performance. The best performer turns out to be based on previously published work with an interesting twist. FIG UK play a significant part in reaching a solution. What more could you ask!

FIG UK member Paul Bennett has a real problem which isn't covered by any textbooks I've seen. The simple technique he mentions for solving integer square roots appeared in Forthwrite not long ago. TIA is short for "Thanks In Advance".

```
From: Paul Bennett peb@amleth.demon.co.uk
Date:

I have a need for a cube root calculation method that works well
with integer maths of either 16 or 32-bit numbers. It is to form
part of a calculation routine for a programme I am constructing.
Despite a search for suitable material in the books I have that
are not in store (still in chaos after the move).

I already know about using logarithm's for the task but this
seemed a little long-winded. I am hoping for something that is as
simple as counting the odd numbers to obtain the square root.

TIA
```

The calculation routine is needed for a problem in hydrodynamics - the flow of fluids.

Our newsgroup readers quickly responded with three very different solutions, two from the USA and one from another FIG UK member. There is only room for a portion of the whole thread here, but the rest can be found on **www.dejanews.com**.

First to appear is a solution from Dr. Julian Noble, author of Scientific Forth, who uses Forth in his training material.

His solution uses the technique invented by Sir Isaac Newton and relies on the mathematical relation[2] that the rate of increase of x^3 at any value of x is

[2] For details, see Jon Bentley, More Programming Pearls pp148-149

3x^2. The initial guess is improved by cycling round a loop until it stops changing.

```
From: Julian V. Noble  jvn@virginia.edu
Date:

Here is a little routine using Newton's method--fairly short &
sweet:

\ Integer cube root
\ version of October 12th, 1999
\ ---------------------------------------------------
\     (c) Copyright 1999  Julian V. Noble.          \
\       Permission is granted by the author to      \
\       use this software for any application pro-  \
\       vided this copyright notice is preserved.   \
\ ---------------------------------------------------

\ This is an ANS Forth program requiring the CORE wordset

: ^2    DUP  *  ;

: guess     ( n -- guess)       \ calculates a first guess
    0 >R   1 SWAP               ( 1 n)
    BEGIN  8 /MOD  ?DUP 0>
    WHILE  SWAP  0>  IF  OVER  R>  +  >R  THEN
           SWAP  2*  SWAP
    REPEAT
    0>  AND  R>  +  ;

: icube_rt  ( n -- root)        \ Newton's method
    DUP  guess                  ( n guess)
    BEGIN   2DUP  ^2  /         ( n guess n/guess^2)
            OVER  2*  +         ( n [n/guess^2]+2*guess)
            3 /                 ( n guess guess')
            TUCK  =
    UNTIL   NIP  ;
```

The second line of attack came from veteran Wil Baden, who offered a solution along the lines suggested by Paul - "counting the odd numbers".

```
Author: Wil Baden <wilbaden@netcom14.netcom.com>
Date: 1999/10/09

The following should do for single numbers when efficiency
is not important.

    : CUBEROOT  ( u -- r )
        >R 0 0      ( root cube -- root )
        BEGIN DUP R@ U< WHILE
```

```
            1 UNDER+
            OVER DUP 1+ * 3 * 1+ +
        REPEAT
        R> DROP DROP ;
```

Whereas "counting the odd numbers" is sufficient for square roots, for cube roots the numbers are more complicated. (For a simpler version of this algorithm, see elsewhere in this issue.)

FIG UK member, Philip Preston, pointed out that there were only a small number of answers for the range of 0 to 65535, so the results could be pre-computed and found when needed using a fast binary search.

```
Author: Philip Preston <philip@preston20.freeserve.co.uk>
Date: 1999/10/10

Hi Paul,

For 16 bit numbers I would construct a table of cubes at compile
time (only 41 cells) and do a binary search on it at run time -
the index of the nearest match to the search key would be its cube
root. The same method would work for 32 bit numbers but the size
of the table would have to increase to 1626 cells which might be
too expensive.
```

Soon afterwards, Philip came back with a better variant still. I am especially pleased that this improved version is based on a previous item in Forthwrite.

```
From: Philip Preston
Date: 1999/10/10


Whoops! I must remember to drink at least two cups of coffee
before posting when I get home on a Saturday night :-)

Of course a table isn't necessary because the cubes can be
calculated on the fly almost as quickly as they could be fetched
from a table. The following is based on Gordon Charlton's
BINSEARCH from Forthwrite #62, October 1991:

\ 40 CONSTANT MAX-CUBE-ROOT ( for 16-bit numbers)
1625 CONSTANT MAX-CUBE-ROOT ( for 32-bit numbers)

: CUBE-ROOT ( u1 -- u2 )
    >R 0 MAX-CUBE-ROOT 1+ BEGIN
        2DUP 1- < WHILE
            2DUP + 2/
            R@ OVER DUP DUP * * U< 0= IF  ( : CUBE DUP DUP * * ; )
                -ROT
            THEN
```

```
                NIP
        REPEAT
        R> 2DROP ;
```

Marcel Hendrix provided a way to compare these offerings on performance.

```
Author: Marcel Hendrix <mhx@iaehv.iae.nl>
Date: 1999/10/10

Here are some measurements (using iForth) that may be interesting
to some:

1000000 VALUE loops

3e 1/F FCONSTANT 1/3
: CUBRTEST ( -- )
CR TIMER-RESET loops 0 ?DO 4294901760 CUBEROOT    DROP  LOOP
.ELAPSED CR TIMER-RESET loops 0 ?DO 4294901760 CUBE-ROOT    DROP
LOOP .ELAPSED CR TIMER-RESET loops 0 ?DO 4294901760 S>F 1/3 F**
FDROP  LOOP .ELAPSED ;

FORTH> cubrtest
5.611 seconds elapsed.    \ Wil Baden's solution
3.952 seconds elapsed.    \ Philip Preston's second solution
5.517 seconds elapsed.    \ Floating point calculation

The co-processor is out-classed (on a P55-166 at least) by
Philip's / Gordon's implementation. If you don't need the extra
precision, don't have the co-processor and when your Forth is not
at least twice faster than iForth it is clear (wrt efficiency and
size) what to do.
```

Paul Bennett acknowledged all these contributions, but we don't yet know which one he adopted.

```
Author: "Paul E. Bennett" <peb@amleth.demon.co.uk>
Date: 1999/10/11

To all those who responded, many many thanks. I have enough to try
out and see what works in my program. I will try and post later
with information about the one I elected to use. Thanks again all.
```

Editor's Note: Forthwrite May '98 included an article from Fred Behringer on integer Cube Roots Without Division. An index with this and other articles back to 1990 can be found on our web-site and is re-printed every January.

Chris Jakeman
cjakeman@bigfoot.com

Cube Roots Again

Chris Jakeman

This item was inspired by the thread on comp.lang.forth (elsewhere in this issue).

I think the algorithm to find an integer square root by counting odd numbers is a classic piece of Forth. It may not be efficient or important, but the algorithm is so simple and neat and makes such good use of +LOOP, that it shines.

Here is the version for square-root, explained in detail in Forthwrite 93 and 94.

```
: SQRT ( square -- root )
    -1             \ Initial Count
    SWAP -1 DO
      2 +          \ Increase Count
    DUP +LOOP      \ Add Count to Loop Index and test >= square?
    2/ ;           \ Extract the result from Count
```

Cube roots are a little more complex, but I found a solution which has a similar approach though with an extra value on the Data Stack. With the square root, we were incrementing the count using `2 +`, with a final `2/` to get the result; with the cube root, we increment using `6 +` and finally divide by 6.

```
: UNDER+ ( n1 n2 n3 -- n1+n3 n2 )     \ As featured in Bons Mots
    ROT + SWAP ;

: CBRT ( cube -- root )
    -6 1         \ initial Count & initial Increment
    ROT -1 DO
      6 UNDER+   \ Increase Count
      OVER +     \ Add Count to Increment
    DUP +LOOP    \ Add Increment to Loop Index and test >= cube?
    DROP         \ Increment
    6 / ;        \ Extract the result from Count
```

Letters

John Hayhow describes a project which pulls together published efforts and makes Forth available on very small, fast micro-controllers. He would be happy to collaborate with other FIG UK members and work has already begun.

This project differs significantly from the F11-UK being run by Jeremy Fowell. F11-UK supports a full Forth system and 64K of RAM whereas John's micro-controllers are cheaper and far smaller. One of the interesting technical options would be the use of multiple communicating micro-controllers, each executing a dedicated task.

John Hayhow

From: jhayhow@freezone.co.uk
Sent: 1 March 2000
To: cjakeman@bigfoot.com
Subject: Forth micro-controller development

Dear Chris,

Many thanks for inviting me to the reunion last November and nudging me into re-joining FIG(UK).

Reading Gordon Charlton's letter in Issue 105 prompted me to revive a project which I have been considering for some time, namely applying FORTH to the programming of micro-controllers.

Micro-controllers such as the Microchip PIC series are being used everywhere, generally not recognised as computer systems but as flexible hardware replacements in a wide range of consumer goods and industrial instruments.

Applying FORTH to these devices raises some significant problems.

1) RAM available is at best a few hundred bytes.
2) Program memory is generally not large enough to hold even a modest FORTH system.
3) Program memory is generally in ROM, requiring programming or re-programming to be done outside the development system.

continued

One of the major benefits of FORTH as a development system is the ability to perform incremental compiling. Instead of the conventional development cycle of Edit-Link-Compile-Test-Groan-Edit_again...etc. it is possible to switch development to COMPILE state (simply by calling a compiling word such as 'colon'), add small
extensions to the application program and immediately test them. The system automatically returns to the INTERPRET state when the compiling words have finished their work.

To apply this technique to micro-controllers would require a development system which would:

1) Partition the FORTH system into a HOST contained in the development platform (a PC or some other computer system) and a TARGET containing the micro-controller hardware. HOST and TARGET will be running on different processors and the HOST will cross compile to the TARGET.
2) Link HOST and TARGET by a serial means which would provide for both communication and switching the TARGET between COMPILING (programming) and INTERPRETING mode.

Until recently this would have been difficult to realise but current new micro-controllers are being designed with FLASH program memory. Erasing memory does not now require placing the device in a UV lamp box. Re-programming can generally be performed "in situ" through a serial link. Switching between erasing, programming, verification and communication can be done by fairly simple hardware which itself uses a micro-controller.

There have been several FORTH programs published on preparing code for programming into micro-controllers.

Tim Hendtlass described a 'Nanocomputer Optimising Target Compiler' (NOTCH) using FPC to prepare code for the Microchip PIC16C84 device. Michael Josefson has written 'F2P' which converts FORTH source to assembler source for

continued

the Microchip MPASM assembler.

Both of these programs use the conventional batch processing method, simply preparing input for a separate assembler-based development system.

Frank Sergeant described a 'Three Word Forth' using the HOST/TARGET approach with the Motorola HC11 as TARGET.

Dave Taliaferro recently wrote a series of articles on 'Remote Target Compiling' in which his end target was the Motorola 50002 DSP device.

Richard Mayer has described a PIC assembler in FORTH and a PIC programmer using FORTH code and the PC parallel port.

All these articles describe in part the means to build a stand-alone development system for micro-controllers. All that is required is to extract the relevant parts and to stitch them together with any additions as required.

It should be possible to design a development system adaptable to any micro-controller device, but there are some devices which would ease the design process and benefit particularly from its application.

Microchip PIC16F84 is the later FLASH version of the widely used PIC16C84. It is a simple low cost device with 68bytes RAM, 1K program memory and 1 1/2 I/O ports.

The articles referred to above are generally concerned with this device and would be an ideal base to develop the hardware link and basic communications.

Microchip 16F877 is one of the upper range PIC devices. It has 368bytes RAM, 8K program memory and 4 1/2 I/O ports, some of which serve alternatively as 10bit A-D converters, timers and serial communications.

continued

This device could be programmed for complex applications such as a stand-alone data logger. It might hold a self-contained FORTH system.

The SX28 device from Scenix Semiconductor uses an instruction set compatible with the PIC devices. It is blazingly fast, the standard part running at 50 Million Instructions Per Second (MIPS).

The SX28 has 137bytes RAM, 2K program memory and 2 1/2 I/O ports. It is intended that all peripheral functions will be realised in software and a number of Virtual Peripherals(TM) are freely available. The serial programming arrangements are different to the PIC devices, making use of the oscillator inputs and avoiding using any of the I/O ports for programming.

The Atmel 89C51 is a FLASH version of the long established MCS8051 device. It is not a RISC processor like the PIC and SX but is widely established in applications. It has 128bytes RAM, 4K program memory and 4 I/O ports, some of which have alternative peripheral functions. This device would be useful in developing expansion of the development system to other processors.

The FORTH system used by the HOST is open to personal choice. My preference is to use HOLON by Wolf Wejgaard as this was originally designed as a HOST for target compiling, has versions for the XX86 and HC11 processors and has now been placed in the public domain.

The choice of FORTH system for the TARGET is limited by the processor memory structure. My preference here is for eForth developed by Bill Muench and intended as a portable system to be easily moved between different processors.

The latest revision by Dr Ting recognises that many applications will be in stand-alone micro-controllers; the FORTH code has been simplified and the code structure changed from direct linked to subroutine linked.

continued

The project is now at the stage described above. I have reviewed the requirements and made some choices. I am now ready to begin constructing a simple TARGET and develop the code required for the TARGET monitor, communications and programming.

A Development System of this performance should have commercial potential. However as much of the ideas have been freely given by the FORTH community and in the FORTH spirit of sharing, any practical results of this project will be placed in the public domain.

If any members are interested in assisting me in this project I would be pleased to hear from them. I can be contacted by email to jhayhow@freezone.co.uk .

The next step in developing Gordon's "automatic house" is to link micro-controllers into a distributed intelligence network. The means to do this have already been described and there are various options for the transmission links. This however is another significant project.

Best regards,

John Hayhow

FIG UK Services to Members

Magazine	Forthwrite is our regular magazine, which has been in publication for more than 100 issues. Most of the contributions come from our own members and Chris Jakeman, the Editor, is always ready to assist new authors wishing to share their experiences of the Forth world.
Library	Our library provides a service unmatched by any other FIG chapter. Not only are all the major books available, but also conference proceedings, back-issues of Forthwrite and also of the magazine of International FIG, Forth Dimensions. The price of a loan is simply the cost of postage out and back.
Web Site	Jack Brien maintains our web site at http://forth.org.uk. He publishes details of FIG UK projects, a regularly-updated Forth News report, indexes to the Forthwrite magazine and the library as well as specialist contributions such as “Build Your Own Forth” and links to other sites. Don’t forget to check out the “FIG UK Hall of Fame”.
IRC	Software for accessing Internet Relay Chat is free and easy to use. FIG UK members (and a few others too) get together on the #FIG UK channel every month. Check Forthwrite for details.
Members	The members are our greatest asset. If you have a problem, don’t struggle in silence - someone will always be able to help. Do consider joining one of our joint projects. Undertaken by informal groups of members, these are very successful and an excellent way to gain both experience and good friends.
Beyond the UK	FIG UK has links with International FIG, the German Forth-Gesellschaft and the Dutch Forth Users Group. Some of our members have multiple memberships and we report progress and special events. FIG UK has attracted a core of overseas members; please ask if you want an accelerated postal delivery for your Forthwrite.