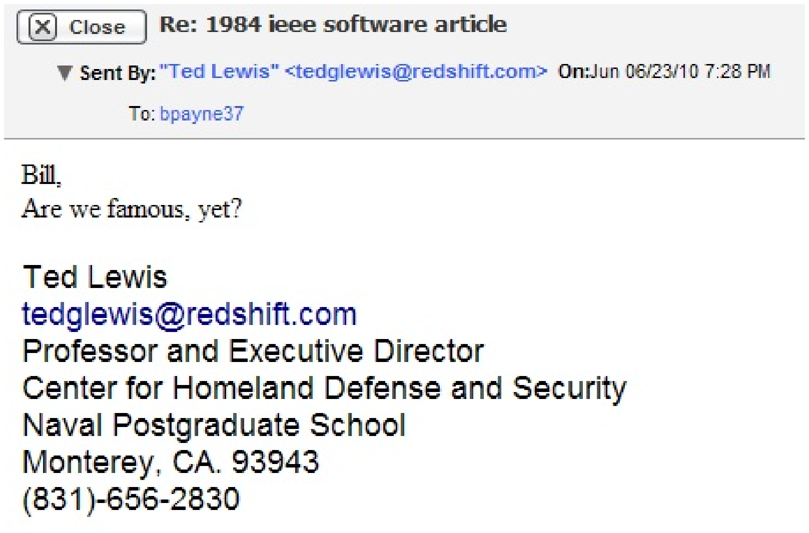
8085 Sandia-Approved FORTH Article

First posted
Monday January 17, 2000 17:29
Updated
Friday June 25, 2010 10:48



Close **Re: 1984 ieee software article**

▼ Sent By: "Ted Lewis" <tedglewis@redshift.com> On: Jun 06/23/10 7:28 PM

To: bpayne37

Bill,
Are we famous, yet?

Ted Lewis
tedglewis@redshift.com
Professor and Executive Director
Center for Homeland Defense and Security
Naval Postgraduate School
Monterey, CA. 93943
(831)-656-2830

Corvallis, Selecting implementation by TG Lewis - 1984 IEEE. Software, 4400 Sulphur Springs Rd., Corvallis, OR 97330. Selecting an implementation ... Payne has responded with the following critique of both Pascal and C. He .... at 9.6 Kbits with an 8085 Forth development system (Figure 2).
ieeexplore.ieee.org/iel5/52/35715/01695234.pdf?arnumber=1695234

IEEE Software, October 1984, Volume 1 Number 4 page 100-102.

ROMable Forth applications code development

Forth programming environment. Forth unlike C, Pascal, and assembler, is a language system. A language system is an amalgamation of an operating system, high-level language, and job control language. A diagram of a minimal Forth microcomputer hardware/software system is shown in Figure 1.

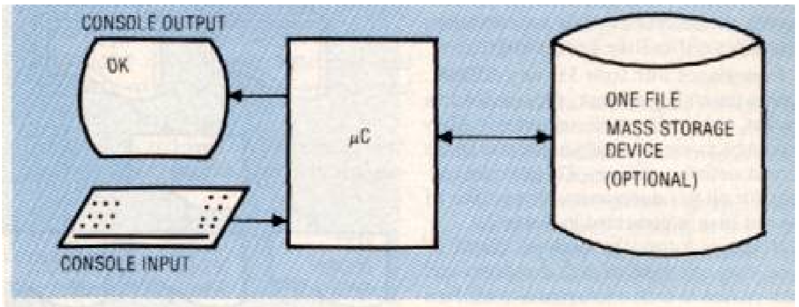


Figure 1. Minimal Forth operating system.

Any terminal can serve as a console and is incorporated into the Forth system by either selecting previously written sub routines or writing new subroutines to do the following: (1) clear the screen and home the cursor, (2) clear screen from cursor to end of line, (3) clear screen from cursor to end of screen, and (4) position cursor at screen coordinates x, y.

Forth programs are incrementally compiled from either the console key board or disk file. High level Forth contains structured control constructs of BEGIN ... UNTIL, BEGIN ... WHILE, CASE ... OF... ENDOF ... ENDCASE, DO ... LOOP.

Forth contains subroutines for number conversion, string processing, I/O for matting, as well as other generic tasks that make high level Forth similar to Basic in terms of the number of services.

Assemblers are frequently written in a language other than assembler, but in the Forth system assemblers are written in high level Forth and assembler routines incrementally compiled. As soon as a subroutine has been assembled, control is returned to the Forth operating system, and the subroutine is available for use by any other program. This "integration" contributes greatly to Forth's portability and speed.

Forth systems incorporate software development tools commonly found in other systems as well, such as full screen editors. Subroutines written in either Forth high level or assembler language interface to other operating systems' file systems and utilities, for example, PC/ MS-DOS and CP/M.

Forth hardware development system.

Any specialized microcomputer system can be transformed into its own Forth development system by adding additional EPROM and RAM to contain a Forth operating system and development applications code. Two asynchronous communications elements provide the terminal and disk ports required by a Forth operating system. The National 8250 asynchronous communications element is attractive for the disk interface because it supports baud rates to 56 Kbits. The Intel 8251 A Usart works well at 9.6 Kbits with an 8085 Forth development system (Figure 2).

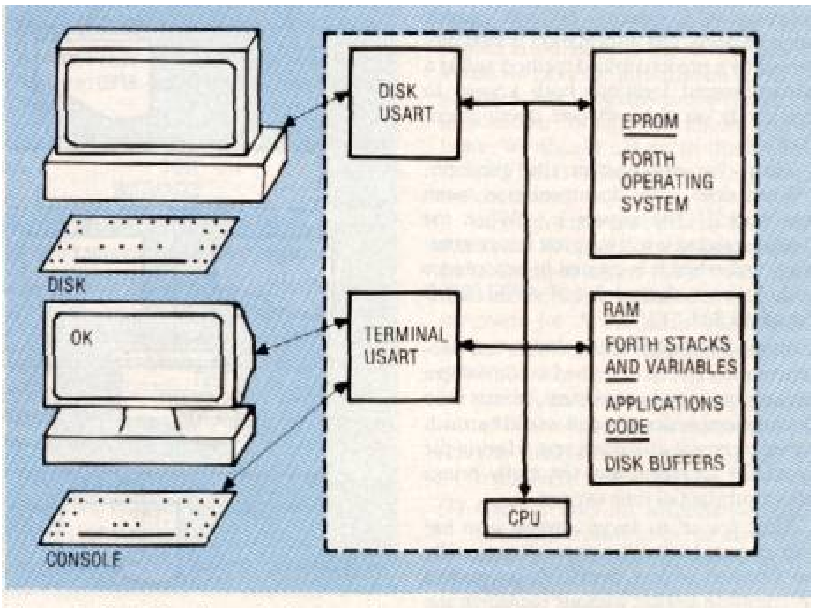


Figure 2. Forth hardware/software microcomputer development system.

Oops - In the download code, the the screen out and disk i/o are multiplexed through the same rs-232 port. It took us a bit of time to realize we should have used multiplexed i/o. Rome was not built in a day.
Saturday January 8, 2000 20:19

The strategy for producing ROMable application code is to keep development code in a disk file on a general purpose microcomputer for compilation onto the target Forth development system. A mini full screen editor on the target Forth development system is used to make small changes in the applications source code while a more powerful full screen editor residing on the general purpose microcomputer is available for extensive editing.

Once the application code is operational, it is added to the target Forth operating system source code. All sub routines in the Forth operating system unreferenced by the application code are ''commented out'' and the resulting minimum sized application source code file is cross compiled onto the target system. A minimum sized, runtime ROMed application binary file results (Figure 3).

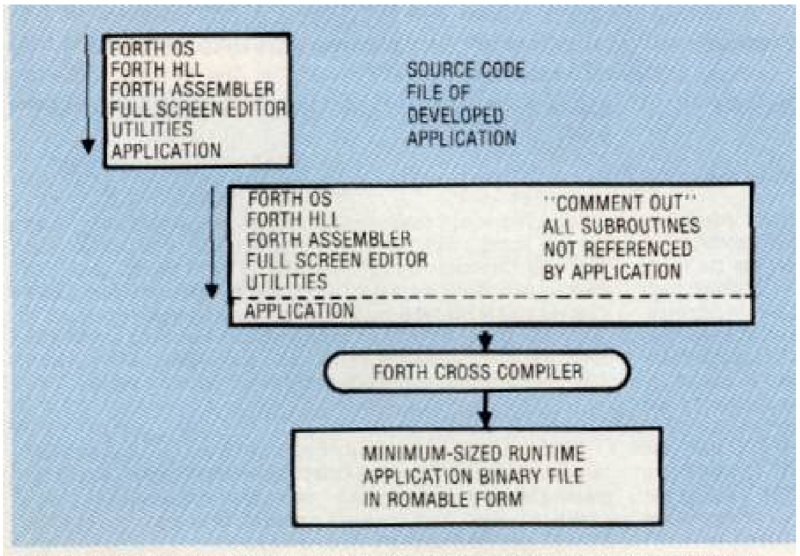

FORTH OS
FORTH HLL
FORTH ASSEMBLER
FULL SCREEN EDITOR
UTILITIES
APPLICATION
SOURCE CODE FILE OF DEVELOPED APPLICATION
FORTH OS
FORTH HLL
FORTH ASSEMBLER
FULL SCREEN EDITOR
UTILITIES
APPLICATION
''COMMENT OUT'' ALL SUBROUTINES NOT REFERENCED BY APPLICATION
FORTH CROSS COMPILER
MINIMUM-SIZED RUNTIME APPLICATION BINARY FILE IN ROMABLE FORM

Figure 3. Final steps to produce minimum-sized ROMable Forth application.

Cost comparisons.

Hardware costs of the Forth approach compared to microcomputer code development, either on supermini computers or microcomputer development systems, is one to two orders of magnitude less. All that is required is a target hardware system with additional EPROM and RAM and two serial I/O ports, a general purpose disk-based microcomputer an RS-232 connected EPROM programmer, and an inexpensive terminal.

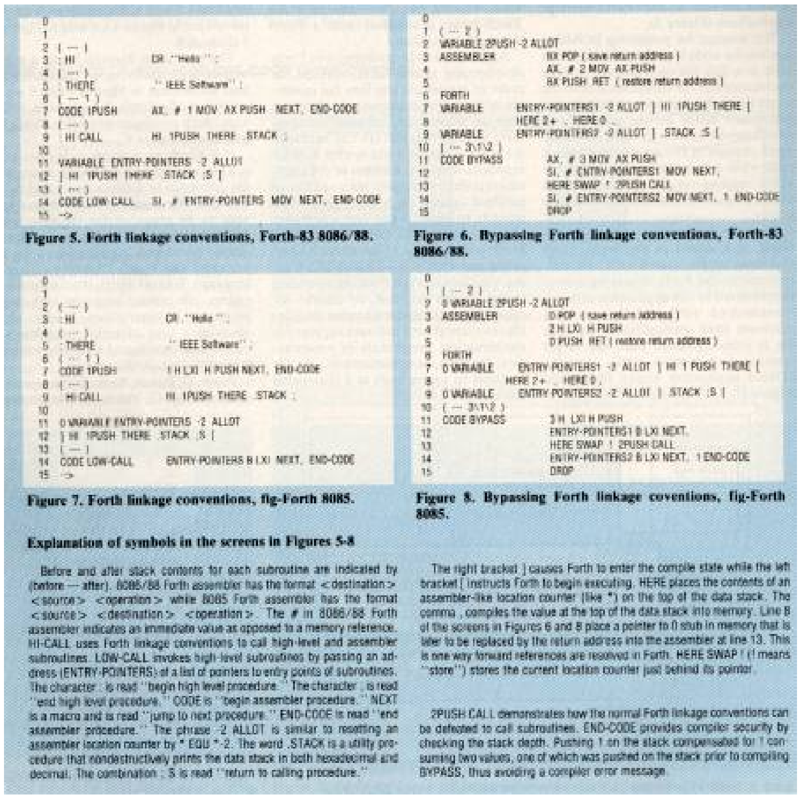
The Forth development system software is an order of magnitude less costly than other systems. A cross compiler and a target source ready to be modified can be purchased for less than $500. No license fees are paid on reselling binaries produced by a cross compiler. All source code, with the possible exception of the Forth base, is supplied with a Forth development system.

Application code developed on a Forth development system takes, perhaps, an order of magnitude less time for moderate or large microcomputer applications than on other microcomputer development systems because: (I) The hardware for a Forth development system is so inexpensive that each member of the application code development team can have a personal system; (2) It takes only a few seconds to modify and test a software change in either assembler or high level Forth code because of incremental compiling and immediate loading and linking. (3) Code is taken from the operating system, Forth language, or utilities for use in the application code rather than being rewritten (Forth software support for mathematical peripherals or coprocessors, graphics, communications, etc., is available in source code at a reasonable cost).

Size and performance of applications code.

Forth produces minimum sized applications code. The reason is that Forth programs are composed of small subroutines. Code is not redundant. Pascal and C, for example, include redundant loop code each time they are coded.

Forth linkage conventions exact about a factor of six speed penalty for high-level Forth applications code compared to the same code written in assembler. However, in many applications most of the work is done in a small part of the code. The formal Forth subroutine linkage conventions can be discarded in favor of simple assembler calls for those parts of the code requiring speed. Examples of invoking high-level and assembler Forth subroutines from Forth and invoking high-level Forth subroutines to produce the output in Figure 4 are seen in Figures 5 through 8.



```
 0
 1
 2 ( --- )
 3 : HI                CR ." Hello " ;
 4 ( --- )
 5 : THERE             ." IEEE Software " ;
 6 ( --- 1 )
 7 CODE 1PUSH          AX, # 1 MOV AX PUSH  NEXT, END-CODE
 8 ( --- )
 9 : HI-CALL           HI 1PUSH THERE .STACK ;
10
11 VARIABLE ENTRY-POINTERS -2 ALLOT
12 ] HI 1PUSH THERE .STACK ;S [
13 ( --- )
14 CODE LOW-CALL   SI, # ENTRY-POINTERS MOV NEXT, END-CODE
15 -->
```

Figure 5. Forth linkage conventions, Forth-83 8086/88.

```
 0
 1 ( --- 2 )
 2 VARIABLE 2PUSH -2 ALLOT
 3 ASSEMBLER               BX POP ( save return address )
 4                         AX, # 2 MOV AX PUSH
 5                         BX PUSH RET ( restore return address )
 6 FORTH
 7 VARIABLE          ENTRY-POINTERS1 -2 ALLOT ] HI 1PUSH THERE [
 8                   HERE 2+ , HERE 0 ,
 9 VARIABLE          ENTRY-POINTERS2 -2 ALLOT ] .STACK ;S [
10 ( --- 3\1\2 )
11 CODE BYPASS             AX, # 3 MOV AX PUSH
12                         SI, # ENTRY-POINTERS1 MOV NEXT,
13                         HERE SWAP ! 2PUSH CALL
14                         SI, # ENTRY-POINTERS2 MOV NEXT, 1 END-CODE
15                         DROP
```

Figure 6. Bypassing Forth linkage conventions, Forth-83 8086/88.

```
 0
 1
 2 ( --- )
 3 : HI                 CR ." Hello " ;
 4 ( --- )
 5 : THERE               ." IEEE Software " ;
 6 ( --- 1 )
 7 CODE 1PUSH           1 H LXI H PUSH NEXT, END-CODE
 8 ( --- )
 9 : HI-CALL            HI 1PUSH THERE .STACK ;
10
11 0 VARIABLE ENTRY-POINTERS -2 ALLOT
12 ] HI 1PUSH THERE .STACK ;S [
13 ( --- )
14 CODE LOW-CALL        ENTRY-POINTERS B LXI NEXT, END-CODE
15 -->
```

Figure 7. Forth linkage conventions, fig-Forth 8085.

```
 0
 1 ( --- 2 )
 2 0 VARIABLE 2PUSH -2 ALLOT
 3 ASSEMBLER               D POP ( save return address )
 4                         2 H LXI H PUSH
 5                         D PUSH RET ( restore return address )
 6 FORTH
 7 0 VARIABLE       ENTRY-POINTERS1 -2 ALLOT ] HI 1 PUSH THERE [
 8                HERE 2+ , HERE 0 ,
 9 0 VARIABLE       ENTRY-POINTERS2 -2 ALLOT ] .STACK ;S [
10 ( --- 3\1\2 )
11 CODE BYPASS              3 H LXI H PUSH
12                          ENTRY-POINTERS1 B LXI NEXT,
13                          HERE SWAP ! 2PUSH CALL
14                          ENTRY-POINTERS2 B LXI NEXT, 1 END-CODE
15                          DROP
```

Figure 8. Bypassing Forth linkage coventions, fig-Forth 8085.

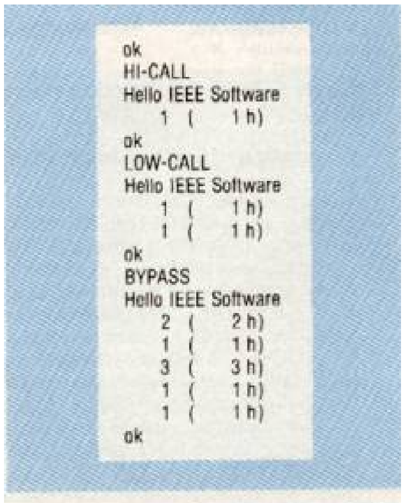
Explanation of symbols in the screens in Figures 5-8

Before and after stack contents for each subroutine are indicated by (before — after). 8086/88 Forth assembler has the format <destination> <source> <operation> while 8085 Forth assembler has the format <source> <destination> <operation>. The # in 8086/88 Forth assembler indicates an immediate value as opposed to a memory reference. HI-CALL uses Forth linkage conventions to call high-level and assembler subroutines. LOW-CALL invokes high-level subroutines by passing an address (ENTRY-POINTERS) of a list of pointers to entry points of subroutines. The character : is read "begin high level procedure." The character ; is read "end high level procedure." CODE is "begin assembler procedure." NEXT is a macro and is read "jump to next procedure." END-CODE is read "end assembler procedure." The phrase -2 ALLOT is similar to resetting an assembler location counter by * EQU *-2. The word .STACK is a utility procedure that nondestructively prints the data stack in both hexadecimal and decimal. The combination ;S is read "return to calling procedure."

The right bracket] causes Forth to enter the compile state while the left bracket [instructs Forth to begin executing. HERE places the contents of an assembler-like location counter (like *) on the top of the data stack. The comma , compiles the value at the top of the data stack into memory. Line 8 of the screens in Figures 6 and 8 place a pointer to 0 stub in memory that is later to be replaced by the return address into the assembler at line 13. This is one way forward references are resolved in Forth. HERE SWAP ! (! means "store") stores the current location counter just behind its pointer.

2PUSH CALL demonstrates how the normal Forth linkage conventions can be defeated to call subroutines. END-CODE provides compiler security by checking the stack depth. Pushing 1 on the stack compensates for 1 consuming two values, one of which was pushed on the stack prior to compiling BYPASS, thus avoiding a compiler error message.

The screens in Figures 5 and 6 were written in Forth-83 for an Intel 8086/88 host while those in Figures 7 and 8 were written in fig-Forth for an Intel 8085 host. Both programs produced the output seen in Figure 4.



```
ok
HI-CALL
Hello IEEE Software
     1  (    1 h)
ok
LOW-CALL
Hello IEEE Software
     1  (    1 h)
     1  (    1 h)
ok
BYPASS
Hello IEEE Software
     2  (    2 h)
     1  (    1 h)
     3  (    3 h)
     1  (    1 h)
     1  (    1 h)
ok
```

Figure 4. Output of programs seen in the screens in Figures 5-8.

If a Forth application code does not run fast enough on a processor, changing the processor to a faster one could be a solution. The assembler code subroutines need to be rewritten for the new processor but most of the high level code remains unchanged. Applications code is perhaps the most portable of any language, because Forth is an operating system, job control language, high-level language, linkage editor, loader, and contains a host assembler capable of macro and conditional assembly written in Forth’s high-level language.

Forth, C, Pascal, Basic, and assembler comparisons. C, Pascal, and assembler are not operating systems. They are not job control languages. Pascal does not contain an interactive assembler.

Basics are frequently operating systems and job control languages. Basics, like Forth, are interactive, but do not have an interactive assembler; they rely on PEEK’s and POKE’s for some hardware interfaces (Forth contains port fetch and store subroutines similar to PEEK and POKE for accessing I/O ports from high-level Forth). Basic application code frequently runs slowly, and Basic code is perhaps the least portable. Basics are more closely related to Forth in philosophy and sometimes in implementation (Sinclair ZX81Basic was partially written in a floating point version of Forth) than most other languages.

Forth was designed as a tool for producing commercially competitive applications code. C and Pascal were not designed for this. While Basics share many of Forth’s attributes, applications written in Basic run too slowly, and it is difficult to cull unused Basic system subroutines to produce a minimum-sized application code.

Assembler lacks the coding and module testing speed of an interactive Forth assembler. Assembler coding can lead to coding modules in assembler that are better coded in a high-level language.

The cost advantage of using the Forth programming environment for producing ROMable applications code for inexpensive microcomputer systems is so great as to make attempts using other operating systems and languages noncompetitive.

W. H. Payne
Digital Subsystems I,
Division 2311
Sandia National Laboratories
P0 Box 5800
Albuquerque, NM 87185

W. H. Payne is currently a project leader at Sandia. He received his BA degree from Whitman College in 1959 and MS and PhD degrees from Purdue University in 1961 and 1964.