A Debugging Environment for Forth

Peter C. Lind and N. Solntseff

Department of Computer Science and Systems
McMaster University
Hamilton, Ontario, Canada L8S 4K1

Abstract

This paper outlines the design and implementation of a visually oriented debugger and execution tracer for Forth. The utility is coded in Turbo Pascal and provides a visual user interface for controlling the execution of a Forth system and a machine-language monitor which performs the context switching between the user interface and Forth and handles a variety of software interrupts.

Introduction

A facility to trace the execution of Forth words is a very useful feature, not only for the examination of error conditions during program development, but also as a help for the understanding of forth implementation details.

Several approaches to the provision of execution traces have appeared in the Forth literature. These may be classified as follows:

(1) Simulation of the high-level Forth code definition to determine the run-time behavior of a word. This approach is described in **[ASP80]**, **[BLA83]**, and **[BRO83]**. The proposal in **[BLA83]** is noteworthy in that it combines word-by-word interpretation with the use of a screen editor. This represents a first step towards a visual Forth environment.

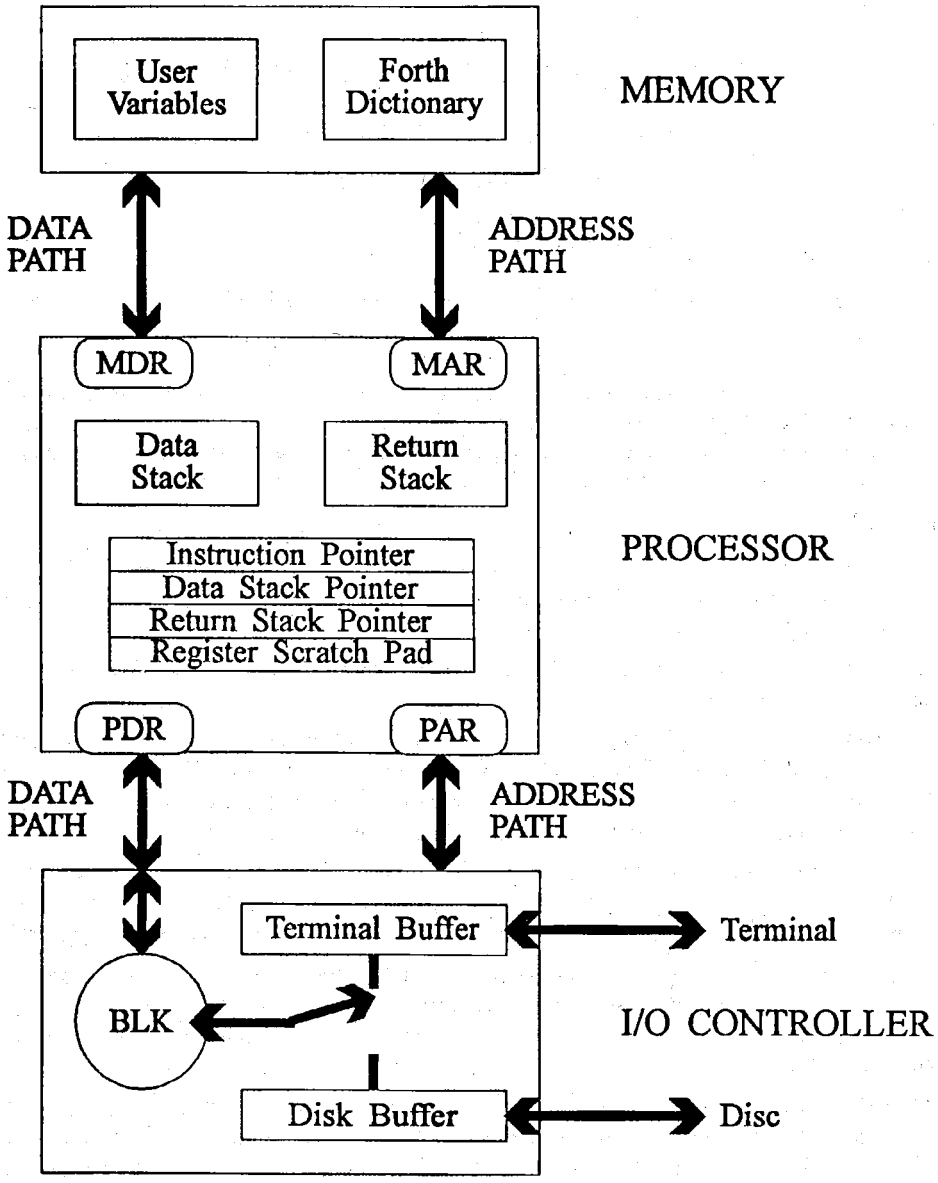
(2) Compilation of calls to a break-point monitor or their definition of `:` and `;` to provide such calls. The former strategy is suggested in **[BRO83]**, **[JOO83]**, and **[VAN81]**, while the latter strategy has been adopted in the highly successful public-domain F83 (**[LAX85]** and **[TIN86]**).

(3) Replacement of an executable address in compiled code by the executable address of a debugging monitor. As this break point can be readily moved at run time from one executable address to the next, single-step operation of the monitor becomes possible.This approach is taken in **[RUS81]** and **[SOL84]**.

None of the above provides for the debugging of words coded in machine language. Yet, as mentioned in the Introduction, Forth implementors need to be able to trace the operation of their systems at the machine-code level. This is especially true when the new system is being generated by meta-compilation, as it is not possible to proceed in an incremental fashion **[TIN86]**. The rest of the paper is devoted to the description of an environment for Forth which includes a complete machine-level debugger.

Specification of TRACER

The TRACER program provides a controlled environment in which a Forth processor may be executed and tested. In addition, TRACER provides an animated tracer/debugger, so that the inner workings of the Forth system can be examined as a Forth word executes; in other words, TRACER is designed as a workbench on which to examine existing Forth kernels or test new ones. The TRACER program has two major components: The *user interface* and the *monitor*. The former is a Pascal program which gives the user control over the debugging environment.



MAR: Memory Address Register
MDR: Memory Data Register
PAR: Peripheral Address Register
PDR: Peripheral Data Register

Figure 1.
The Architecture of the Abstract Forth Machine.

The latter, written in assembly language, handles context switching between Forth and the user interface. The monitor manages all exceptions generated by the Forth system.

Although the work described in this paper was done in an MS-DOS environment with a particular implementation of Forth **[LAX84]**, the design of the debugging environment was performed in as implementation-independent a manner as possible, on the basis of the concept of an abstract forth machine. As shown in Figure 1, a Forth system can be described in terms of an abstract machine **[SOL82]** consisting of a cpu with four registers, two stacks, memory, and input and output streams. In the following, this will be referred to as the Abstract Forth Machine (AFM).

This section describes the requirements of the *monitor*, the program that forms the major portion of the debugging environment. The purpose of the program can be summarized as follows:

(1) Load an executable Forth processor module from disk and, if necessary, perform relocation; establish the correspondence between the AFM and host-machine CPU registers;

(2) Determine the addresses of all Forth primitive interpreters, namely, those implementing **`constant`**, **`variable`**, **`vocabulary`**, as well as those created by means of the **`create ... does>`** mechanism;

(3) Establish the structure of the vocabulary used by the Forth implementation being studied;

(4) Make available a high-level Forth de-compiler, as well as a low-level Forth dis-assembler.

Implementation of TRACER

The TRACER program is an animated tracer/debugger. The internal workings of Forth are displayed as execution proceeds and the user can gain a bird's eye view of the entire system. In other words, the prime goal in the design and implementation of TRACER is the provision of a visual interface to Forth giving the user insight into its innermost mechanisms.

The first action of the TRACER program (see Block 1 of Figure 2) is to load a Forth kernel description which consists of (1) the mapping between the AFM and host-machine cpu registers, (2) the host-machine addresses corresponding to the primitive interpreters listed above, as well as the more important constants such as **`context`**, **`dp`**, **`sp0`**, **`r@`**, etc., (3) the sizes of the various fields (code, link, name, and view fields) in the words compiled into the Forth dictionary, (4) the structure of the Forth dictionary, including the number of threads used in the vocabulary implementation, and, (5) the name of the Forth executable file. This information allows TRACER to assume control over the execution of the Forth system which it does by opening the Forth file, allocating a memory area to the executable file, and, finally, loading the Forth system and relocating it, if necessary.

The second action of TRACER (Block 2) is to place a breakpoint trap at the start of the Forth inner interpreter (usually implemented as a few host-machine instructions) and transfer control to the Forth cold-boot routine. This generally initializes the Forth system, displays an initial message, and transfers control to the outer or text interpreter. Before this can happen, however, the breakpoint previously set will allow TRACER to regain control and record the values of important variables such as **`sp0`** and **`rp0`**. Figure 3 shows the TRACER screen immediately before the startup banner is displayed.

The last action of TRACER (Block 3) is to display a menu and await input from the user. A picture of the initial menu is shown in Fig. 3; here, the top half of the screen shows: The AFM and host-machine registers, the contents of several cells at the top of both data and return stacks, and the contents of the Forth input buffer. Console output from the Forth system is sent to the bottom half of the screen. The list of menus is displayed at the top of the screen. The available options are discussed next.

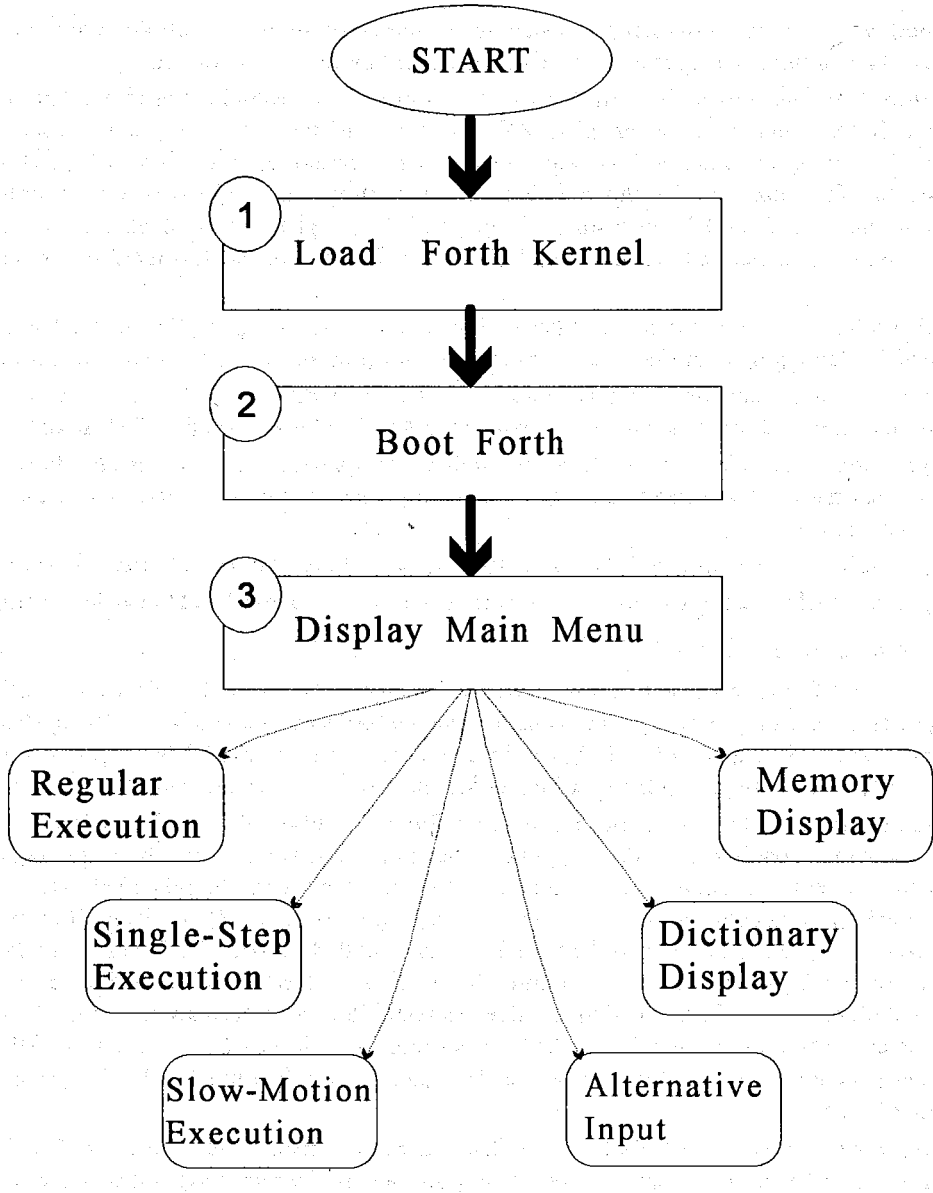


Figure 2.
A General View of the TRACER Options.

Available Options

The options available from the top-level menus shown in Fig. 3 are described in greater detail in this section in order of their appearance on the console screen.

1. System Menu.

(a) Kernel: Edit Kernel Description. This option allows the user to edit the kernel description mentioned in the preceding section. The invocation of this option results in the pop-up window

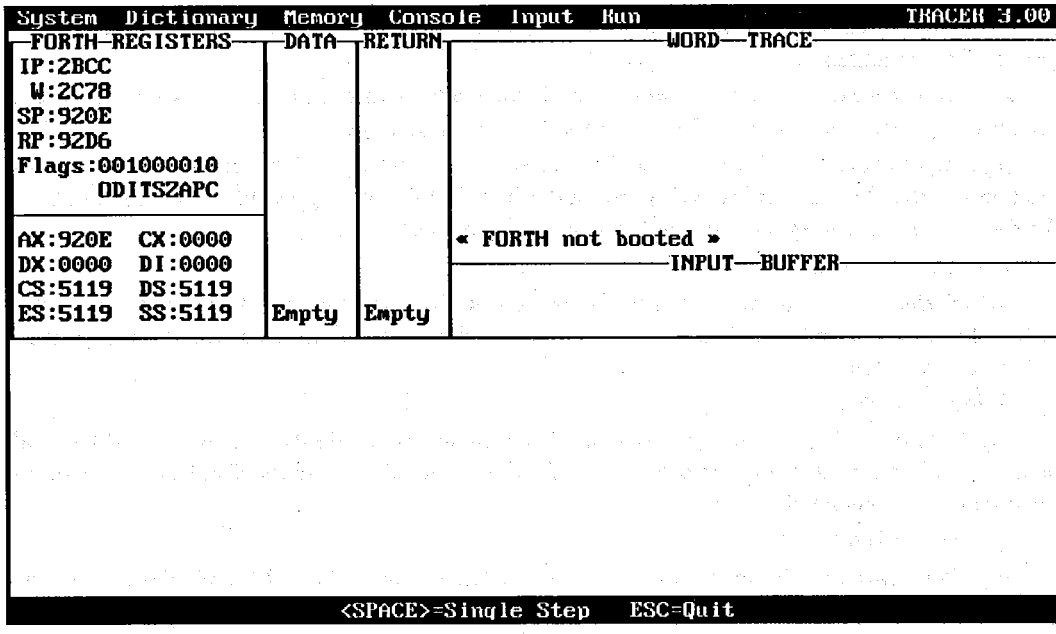


Figure 3.
The Main Display of the TRACER Program.

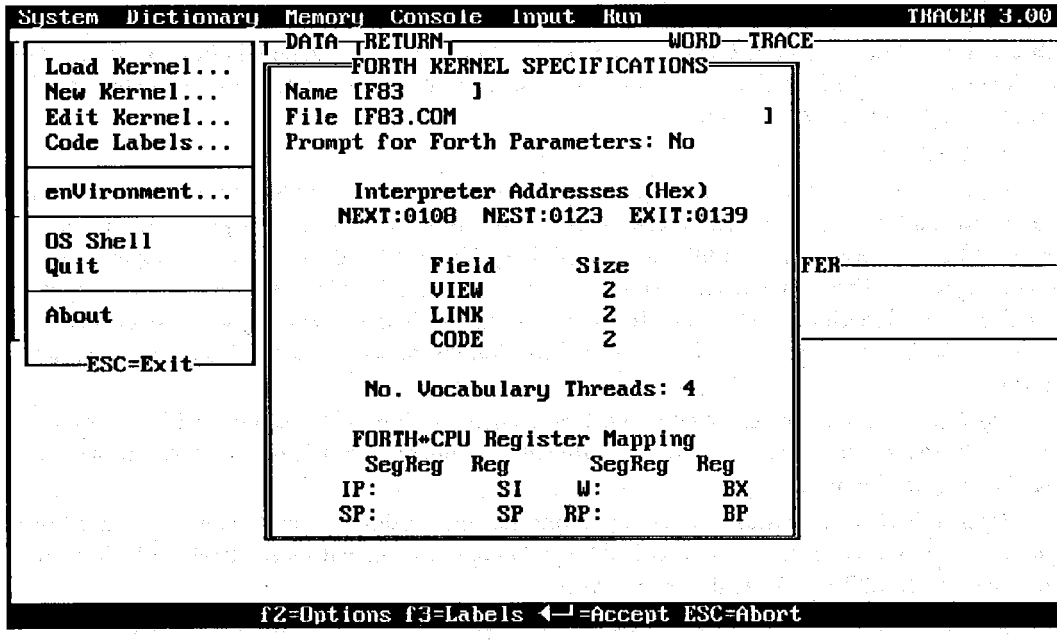


Figure 4.
The Pop-Up Screen for Editing the
Abstract Forth Machine Parameters.

shown in Fig. 4. All of the fields shown there can be changed by the user to tailor TRACER to a specific implementation.

(b) Shell: Invoke Operating System. This is the standard way of temporarily exiting to the operating system to invoke an OS utility, or call a user program.

(c) Environment: Display Vocabularies and System Parameters. This option is not shown on the menu screen because of lack of space on the bottom line, but a press of the E key results in the display of some frequently consulted Forth-system parameters.

2. Dictionary Menu.

(a) Dictionary: Decompiler/Disassembler Option. This is the decompiler and disassembler option that can be used to obtain a dictionary oriented display of individual Forth words—be they colon or code words.

3. Memory Menu.

(a) Memory: Display Memory Contents. This menu option yields a display in ASCII and hexadecimal of the memory area used by the Forth system. The contents of individual memory locations can be changed.

4. Console Menu.

(a) Zoom In/Out. This allows the user to switch between the TRACER screen and the conventional Forth output screen.

(b) Options: Modify AFM Display Updating. Here, the user is allowed to specify which sub-windows of the AFM display shown in Fig. 3 are to be updated during auto-execution of TRACER.

5. Input Menu.

(a) Input: Alternative-Input Option. Here, the Forth input stream is switched from the standard terminal input buffer to a separate buffer whose contents can be edited as needed. The alternative buffer is displayed on the screen and used by TRACER to supply input to Forth. The buffer can be refilled by the user when it empties. Input can be switched back to the regular terminal input buffer at any time.

6. Run Menu.

(a) Go: Regular Execution from Specified Address. Here, Forth is allowed to execute without any constraints starting from an address entered by the user when requested by a pop up secondary menu which requests a sixteen-bit value for (CS,IP). The TRACER program can regain control, however, when (1) a user-placed breakpoint is encountered, (2) the user presses a user-defined break-key combination which results in a forced transfer of control even if no break points have been set (a feature used to escape from infinite loops), and (3) the Forth system terminates normally.

(b) Run: Resume Normal Execution. Here, Forth is allowed to execute without any constraints from the current address, i.e., the value of IP at the point at which the TRACER program has last regained control.

(c) Single-Step Execution. As the name implies, this option is used to step through a Forth program one word at a time. It is invoked by pressing the space bar on the keyboard. Control is returned to TRACER at the end of every word.

(d) Auto: Slow-Motion Execution. In this mode of operation, there is a user adjustable pause at the completion of every Forth word. As in the case of regular Forth execution, the TRACER program is re-entered whenever the user presses the break-key combination, a user-placed breakpoint is encountered, or the Forth program terminates normally. In addition, the pressing of any key during slow-motion execution also allows TRACER to regain control.

The next section deals with the operations performed by TRACER when it is given control from any of the main-menu options.

Context Switching Between TRACER and Forth

Figure 5 shows a block diagram of the TRACER module that handles the three separate execution modes specified through the main menu. Each of the blocks in Fig. 5 is discussed below.

1. Initialize TRACER. The main purpose of this block is to place breakpoints at the start of the NEXT, NEST, and UNNEST interpreters, so that control can be returned to TRACER at these critical stages of the AFM operation. This block is also used to install any user-specified breakpoints.

2. Call Machine-Code Interface. Call external procedure as entry point to machine code interface.

3. Save TRACER Registers. The cpu registers relevant to the operation of TRACER are saved. These include the hardware stack register.

4. Set I/O Traps. All changes to the interrupt vectors required to install both user breakpoints and the user-defined break-key combination are made at this stage.

5. Restore Forth Registers. This block takes care of the placing of the appropriate values into all AFM registers (including the Forth stack pointers). These values are the result of the initialization process shown in Fig. 2 or of a previous invocation of the Save Forth Registers (Block 8 in Fig. 5).

6. Display Machine State. Here, the display of the AFM and cpu registers, the data and return stacks, the current input buffer, and the current Forth word is updated on the screen.

7. Transfer to Forth. Normal Forth execution is resumed at this stage through the execution of a *return-from-interrupt* instruction.

Once Forth execution is re-started as indicated in the last operation, it will proceed autonomously until interrupted by a breakpoint event or a trapped Forth I/O request. The former maybe caused by (1) a user-inserted trap, (2) a trap that implements single-step or slow-motion execution, (3) the normal termination of the Forth program, or (4) a hot-key interrupt.

The course of TRACER actions in the case of a breakpoint event is as follows:

8. Save Forth Registers. All AFM registers are saved.

9. Remove I/O Traps. The changes to the interrupt vectors made before transferring to the Forth system are undone.

10. Restore TRACER Registers. The register contents appropriate to TRACER operation (including the hardware-stack register) are restored.

11. Slow-Motion-Execution Control. If slow-motion mode is in effect, the cycle shown in Fig. 5 is repeated for the next Forth word after a user-specified delay. Otherwise, control is transferred back to the routine which handles the main menu.

The sequence of actions in the case of a trapped Forth I/O event is the following:

12. Service I/O Request. In the case of an input request, the next character is supplied from the TRACER input buffer when TRACER is trapping input, otherwise it is taken from the Forth Terminal Input Buffer. When Forth sends a character to the display screen, the Forth output area is scrolled when becomes necessary.

13. User-Initiated Break. When the I/O request involves the break-key combination (hot key depressed), processing continues as if this were a breakpoint event (see Fig. 5).

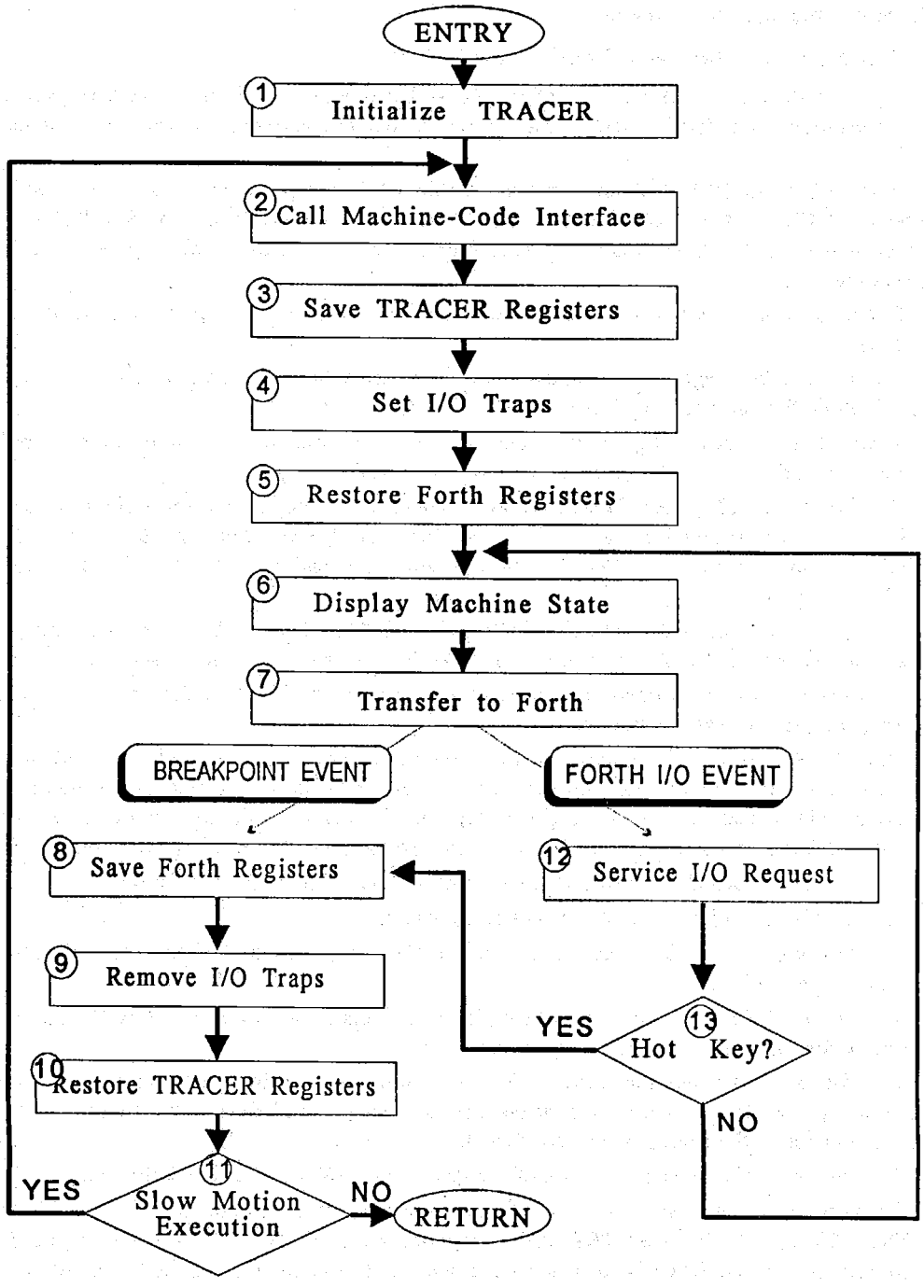


Figure 5.
Details of Context Switching Between TRACER and Forth.

Details of Operation

Figures 6 to 13 provide a sampling of the capabilities of our debugging system. After the system is booted and is executing the outer interpreter, a simple colon definition is entered by means of the Alternative-Input Option as shown in Fig. 6. It should be noted that trapping of

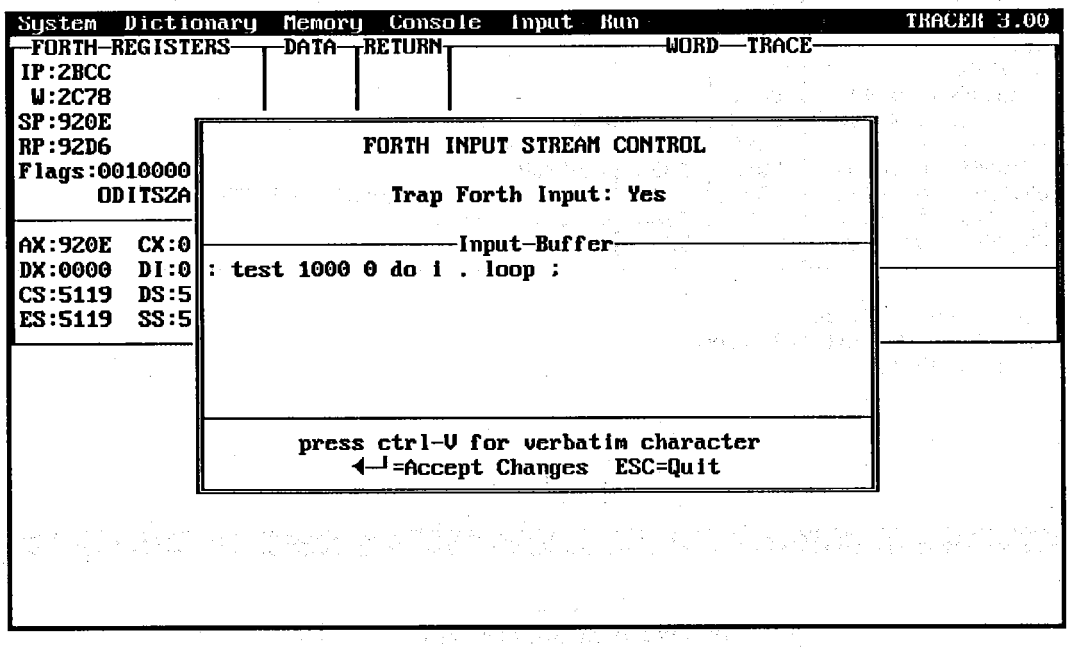


Figure 6.
Use of the Alternative-Input Buffer by Forth.

Forth input is enabled, so that when Forth execution resumes, keyboard input will be taken from the TRACER Input Buffer. The Slow-Motion Option is used to resume Forth execution which results in the alternative input buffer being read by the outer interpreter. Figure 7 shows the compilation process interrupted by the operation of the user-defined break key after the parameters of the **loop** have been read.

The remaining figures illustrate the exploratory features of TRACER which become available when regular Forth execution is suspended in the middle of a colon definition (see Fig. 7). Figure 8 shows the window displayed when the environment-parameter display is selected. Note that the system is in the compile state (**state=0**). The manner in which the Forth dictionary can be examined is depicted in Fig. 9. Only one thread of a vocabulary can be displayed at a time. Immediate words are marked by the symbol I in the fourth column. The highlighted word in the display (**DUMP** in Fig. 9) can be decompiled by pressing the enter key. Cursor keys are used for selection of the word to decompile. The decompiler creates a new window as shown in Fig. 10. The decompiler recognizes different classes of Forth words, including words defined by the **create ... does>** mechanism and code words (see Fig. 11).

Lastly, Figures 12 and 13 illustrate the use of the Memory-Display Option. The former exhibits the hexadecimal and ASCII memory dump of the lowest area of the F83 system. Figure13 shows the corresponding disassembly. The cold and warm boot transfers at offsets 0100 and 0103

can be seen, as well as the instructions implementing the Forth inner interpreter (offsets 0106 to 010C, respectively).

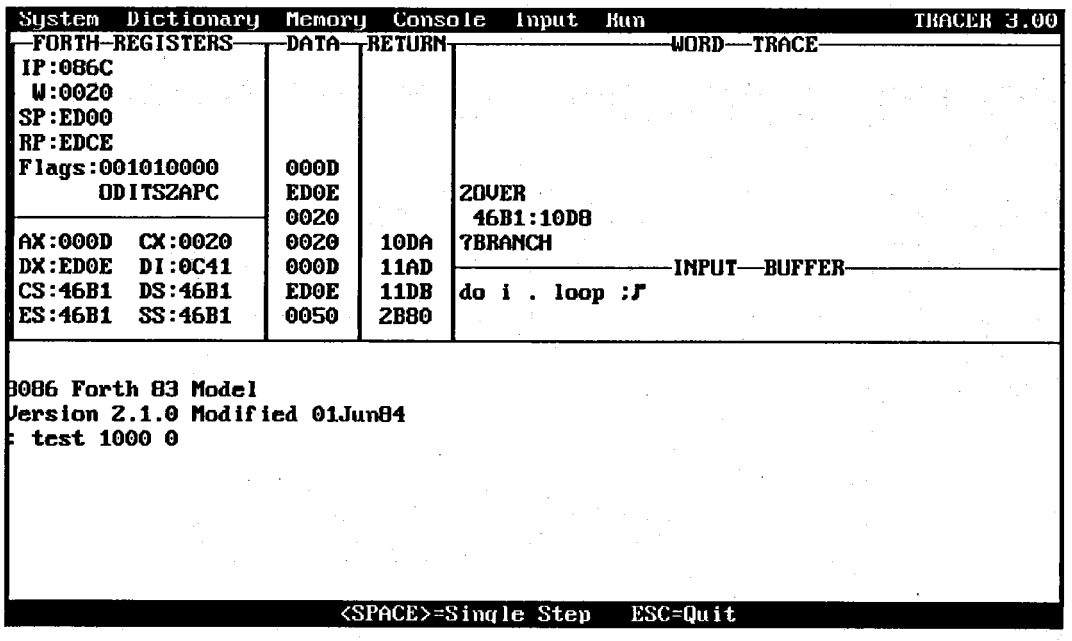


Figure 7.
A Break-Key Interruption of a Forth Compilation.

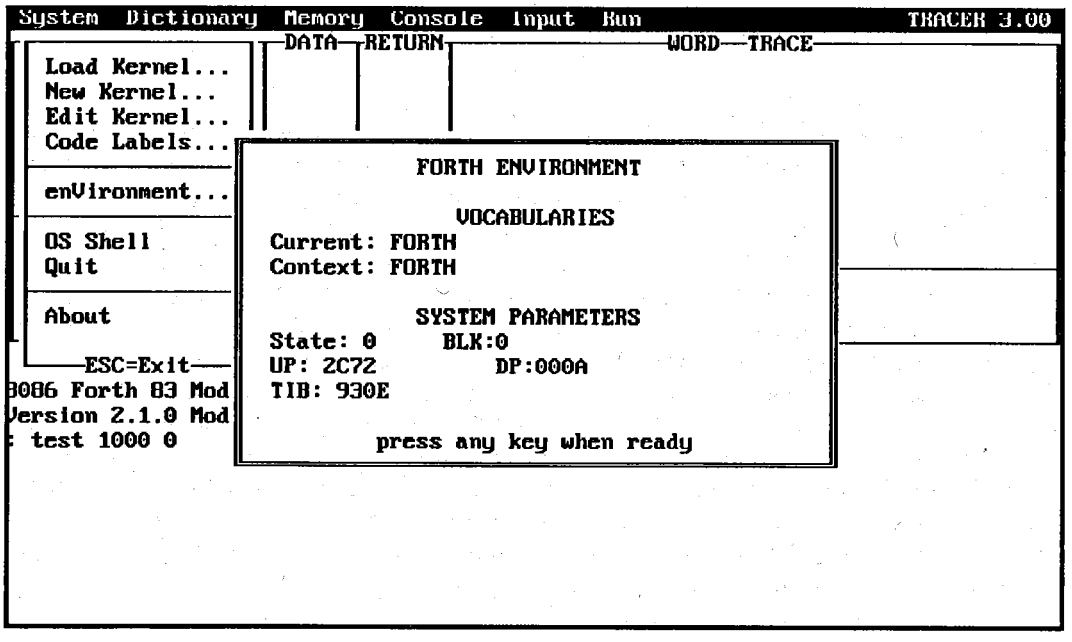


Figure 8.
Display of the Forth Environment during an Interrupted Compilation.

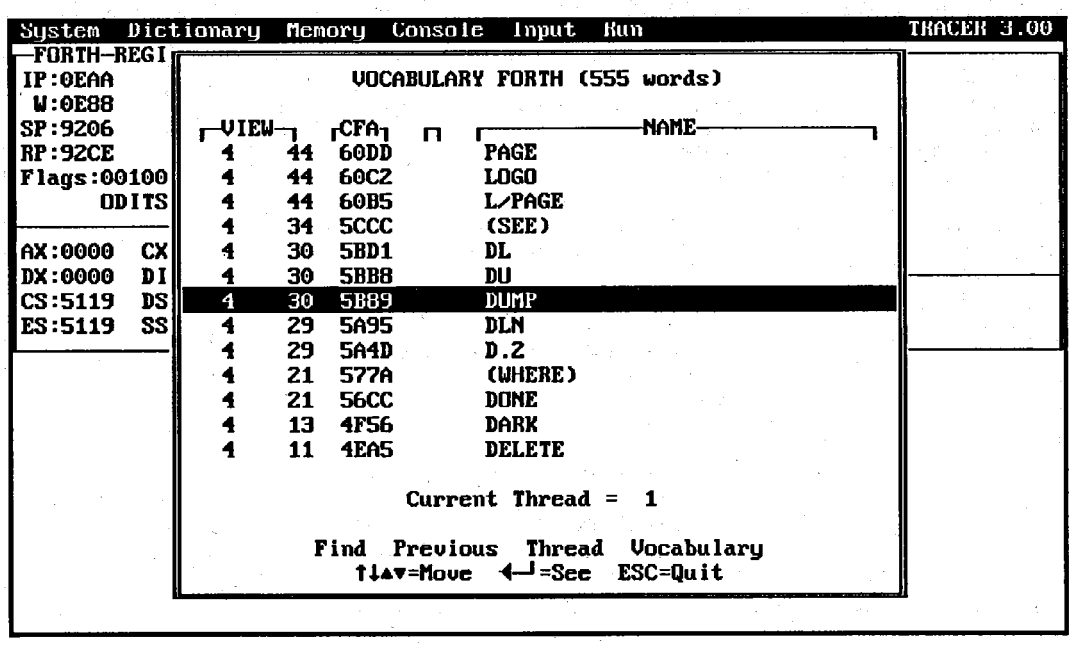


Figure 9.
Display of the a Colon-Definition Decompilation during an Interrupted Compilation.

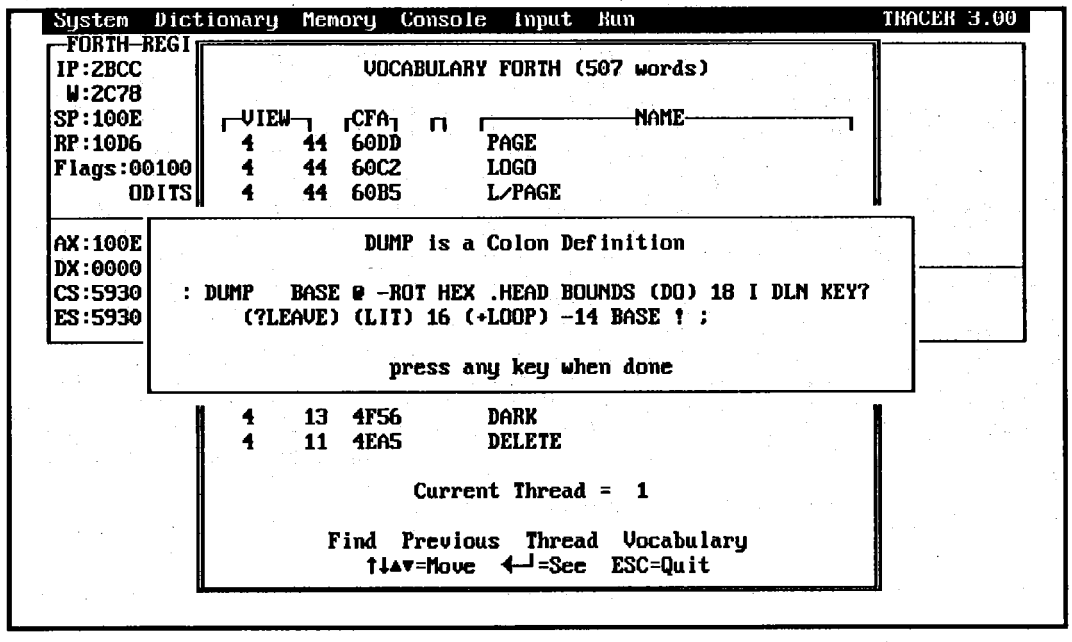


Figure 10.
Display of a Colon-Definition Decompilation
during an Interrupted Compilation.

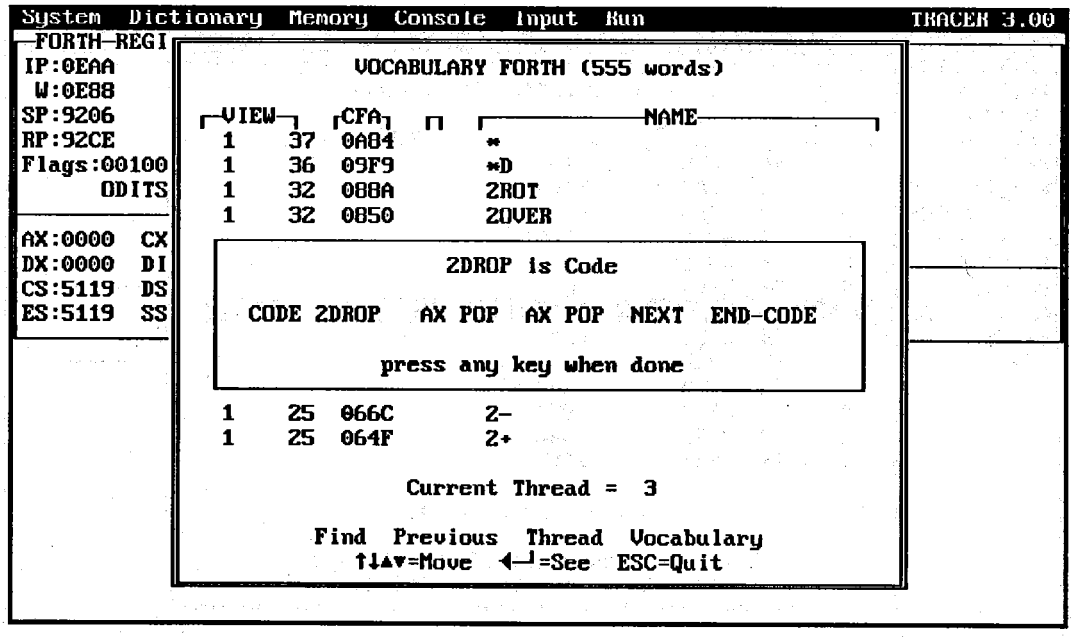


Figure 11.
Display of a Code-Word Disassembly during an Interrupted Compilation.

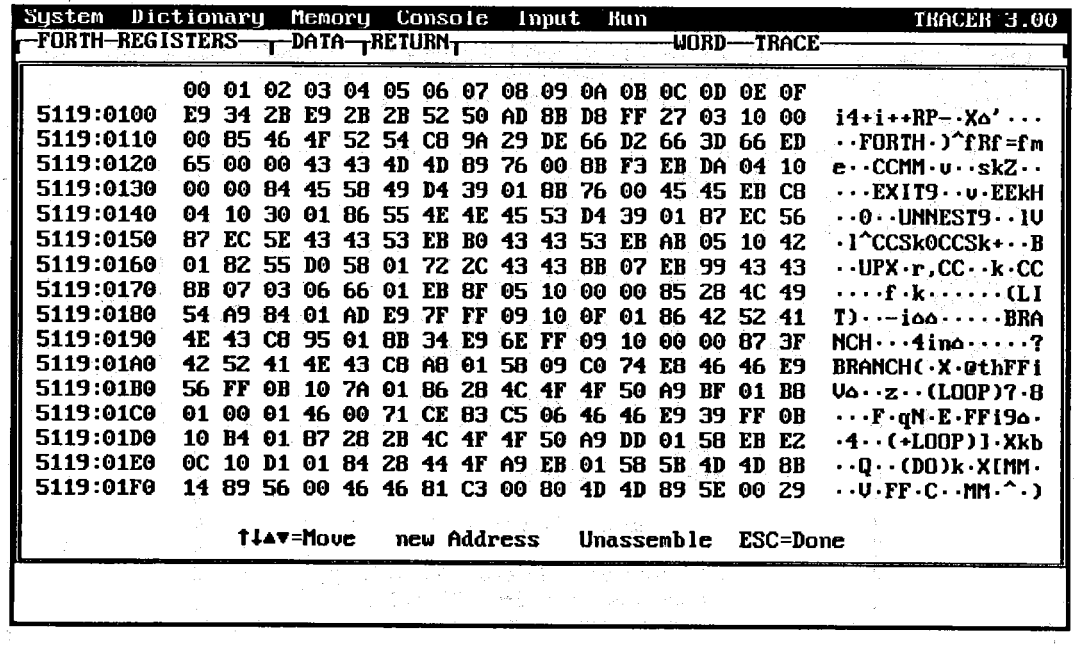


Figure 12.
Display of Memory Contents during an Interrupted Compilation.

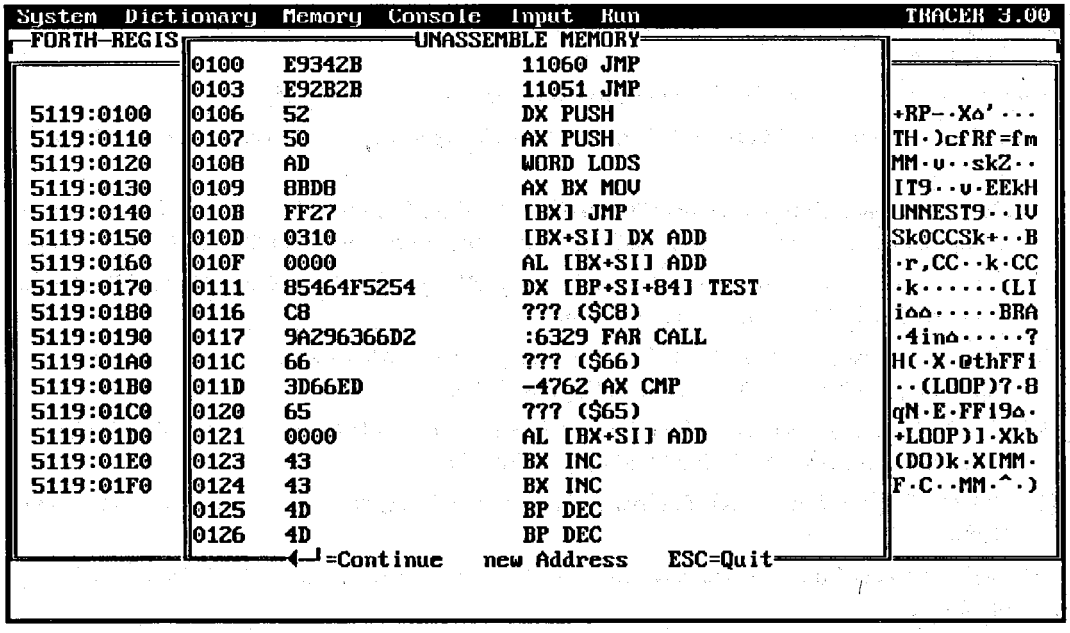


Figure 13.
Display of Disassembled Memory Contents
during an Interrupted Compilation.

Discussion and Conclusions

The TRACER program is an animated tracer/debugger. The internal workings of Forth are displayed as execution proceeds and the user can gain a bird's eye view of the entire system. In other words, the prime goal in the design and implementation of TRACER was the provision of a visual interface to Forth giving the user insight into its innermost mechanisms. The use of TRACER in a graduate course on Forth is planned for the 1991/92 Academic Year. In addition, TRACER has proved to be of considerable value in testing new versions of Forth obtained through metacompilation **[Lax85]**. At present, TRACER works well with F83 and all functions of the program have been implemented. TRACER has been designed to work equally well with 16-bit or 32-bit addressing models, although the latter feature has not been tested. A desirable feature which has not yet been implemented is a facility for user-inserted breakpoints in Forth words. This will be included in the next version of TRACER.

References

[ASP80] T. Asprey, "A Forth execution simulator for debugging, "*Proc. 1980 FORML Conference*, Forth Interest Group, San Carlos, CA94070 (1980), pp. 181-187.

[BLA83] T. Blakeslee, "Debugging from a full-screen editor," *FORTH Dimensions*, V, No. 2 (July/August 1983), p. 30.

[BRO83] L. Brodie, "Add a break point tool," *FORTH Dimensions*, V, No. 1 (May/June 1983), p. 19.

[COl81] B. A. Cole, "A stack diagram utility," *FORTH Dimensions*, III, No. 1 (May/June 1981), pp. 23-32.

[JOO83] R. Joosten, "Tracer for colon definitions," *FORTH Dimensions*, V, No. 2 (July/August 1983), pp. 17-18.

[LAX84] H. Laxen and M. Perry, Public Domain distribution of F83(1984).

[LAX85] H. Laxen and M. Perry, *F83 Source*, Offete Enterprises,Inc., San Mateo, CA 94402 (1985), 209 pp.

[LIN88] P. C. Lind, *Towards Object Oriented Forth*, M.Sc.(Computation) Thesis, Department of Computer Science and Systems,McMaster University, Hamilton, ON L8S 4K1 (1988), 206 pp.

[RUS81] T. Rust, "Terse debugging package," *Proc. 1981 Rochester Forth Conf.*, Univ. of Rochester, Rochester, N.Y. 14623 (1981),pp. 373-374.

[SOL82] N. Solntseff, "An abstract Forth machine," *Proc. 1982Rochester Forth Conf.*, Univ. of Rochester, Rochester, N.Y. 14623(1982), pp. 149-156.

[SOL84] N. Solntseff, "A break point utility for Forth," *Proc. 1981 Rochester Forth Conf.*, Univ. of Rochester, Rochester, N.Y.14623 (1984), pp. 373-374.

[TIN86] C. H. Ting, *Inside F83, Revised Edition*, OffeteEnterprises, Inc., San Mateo, CA 94402 (1986).

[VAN81] P. van der Eijk, "A stack diagram utility," *FORTHDimensions*, III, No. 1 (May/June 1981), pp. 23-32.