# An Implementation of MC68000 fig–Forth under UNIX

P.R. Blake and N. Solntseff
Department of Computer Science and Systems
McMaster University
Hamilton, Ontario L8S 4K1

## 1. Introduction

The Department of Mathematical Sciences (the previous home of Computer Science at McMaster) acquired three Pixel–80 computers in January 1984 for graduate and senior undergraduate teaching. The computers incorporate a MC68000 processor and are operate under the Unix [1] operating system. Fig–Forth has been available on a variety of eight–bit microcomputers for three years and has been used in several undergraduate projects. A graduate course –– CS718: Scientific Applications of Personal Computers contains an introduction to Forth as a real–time system. Prior to 1984, CS718 was run using the microcomputer laboratory equipped with a number of MCS6502 computers. It was decided to transfer this course to the new machines during the summer of 1984.

The present paper summarizes the work done to implement Forth on the Pixel–80s. The version chosen was fig–Forth and the implementation of the kernel was done in machine code. Section 2 discusses the impact of the 32–bit architecture of the MC68000 on the 16–bit Forth model. Section 3 outlines the manner in which C–coded routines can be used to implement new Forth primitives. Section 4 summarizes the extensions to the Forth vocabulary that were needed to implement string operations and variables, file management in conjunction with Unix, and an interface to the standard Unix tools. Finally, Section 5 contains a brief assessment of the project. A more detailed account of this work can be found elsewhere [5].

## 2. The Model

The selection of fig–Forth as the version to implement was dictated by the use of this version in the departmental Microcomputer Laboratory. Several unpublished student projects have previously explored the implementation of fig–Forth in C, Fortran, and Pascal. The overall conclusion reached was that high–level languages were not very suitable for implementing Forth with its attributes of easy extensibility and free access to the host's machine language. Because of these requirements, it was decided to implement the Forth kernel in 68000 machine language. Considerable effort was saved through the availability of an implementation under the CP/M–68K operating system kindly provided by Christoph K. Kukulies [2].

The addresses, data paths, and stacks of the abstract Forth machines (AFM) [3] underlying Fig–Forth, as well as Forth–79 and Forth–83 standard models, are all sixteen bits wide. This means that without special measures,

an implementation such as that received from [2] is meant to reside in a 64–Kbyte segment of a MC68000 computer. This is an intolerable restriction in the case of a multi–user operating system such as Unix [1], where a loadable program must be in the relocatable format needed by the operating system. It was decided to retain the AFM architecture of 16–bit addresses and values, but to treat this as an embedding in a 32–bit world of Unix as implemented on a MC68000 computer [4].

The solution adopted was to consider AFM addresses as 16–bit offsets from a 32–bit base address which represents the start of a load module. In order to convert the contents of the Forth instruction pointer (IP), say, which is a 16–bit offset into a 32–bit effective address, the former has to be loaded into a 32–bit scratch register, with its upper half cleared to avoid sign extension. The base address is then added and the result moved into an address register for further use. The conversion of a 32–bit effective address into a 16–bit AFM address is done in an analogous manner. Details of the implementation can be found in [5].

**3. Interface to the C Language**

Apart from a very small nucleus implemented in machine language, the whole of the Unix operating system is written in the C language. Thus, the C/Unix interface provides the necessary mechanisms for program access to the filing system, device I/O, and the Unix shell. Because of this, the Forth/Unix interface was also designed as a set of C–language routines. Moreover, this approach was extended to new Forth primitives that would otherwise have been coded in machine language. The Script facility in Unix is a valuable feature which allows the user to keep a complete record of an interactive session. As one of the significant attributes of Forth is its high degree of interactivity, it was decided to include a script–like feature in the new implementation.

The main problem in interfacing C and Forth routines lies in the way the two languages use their stacks. The C stack is found at the top of virtual memory, namely, $FF,FFFF in hex, and is 32 bits wide. The Forth stack is to be found within the Forth 64–Kbyte segment and is, of course, 16 bits wide. The difference between the base addresses of the Forth segment (approximately $20,0000 in our system) and the C stack is greater than a 16–bit offset can accomodate. Thus, the Forth system has to be aware of the existence of the C–stack. The Forth model as received [2] makes use of all eight address registers, so that it is necessary to exchange stack registers when moving from Forth to C and vice versa.

The following steps have to be taken within a Forth primitive which calls a C procedure or function:

(1) switch return stack pointers,

(2) push any arguments passed to the C routine onto the C stack, after extension of 16–bit addresses and values to 32 bits,
(3) save all Forth registers, call the C routine, and afterwards restore the Forth registers,
(4) if there is a value returned by the C routine, truncate it to 16 bits and leave on the Forth data stack,
(5) transfer control to the Forth address interpreter NEXT.

It should be noted that machine code to perform steps (1) to (5) above has to appear in every Forth word that calls a C routine.

**4. Extensions to Forth**

The new implementation contains several new words that that were included to meet the requirements outlined in the introduction. They fall into three major categories: string extensions, file–management words, and general Unix interface tools. The extensions are briefly discussed below (a more detailed account can be found elsewhere [5]).

String extensions have been adapted from those considered in [6] and include string literals, string variables, and a variety of operators. Run–time checks are included whenever necessary to ensure that string stores are within bounds imposed by the allotted string space.

File–management extensions were directly patterned on the words introduced in [7]. Fig–Forth as implemented on the Pixel–80, can only access its own files which are of type *screens*. A Unix editor can be used to edit a screen file, but only after it has been converted to the standard Unix format. Any Forth editor can be used for editting a screens–type file. The access words *read*, *write*, and *modify* are the same as described in [7]. Most of the remaining file words (*active*, *close*, *default*, *delete*, *file*, *include*, *make*, and *open*), taken from [7], were implemented in a different manner in order to take account of Unix–system features. In addition, several new words were also included to improve access to the filing system. A more detailed account of this is given in [5].

The Unix interface tools have been added to the system to provide a number of convenient procedures for performing several frequently needed tasks. These are: (1) *" <any Unix command string>" unix* to invoke a Unix command; (2) *" <file name>" scr1 scr2 lpr* to print a range of screens from the specified file; (3) *on script* will result in the compilation of a complete history of an interactive session, *print script* will send the script file to the printer, *off script* will stop the accumulation of dialogue on the script file, while *clear script* will erase the script file; finally, (4) *" <file name>" save–forth* will create a new loadable module containing the original Forth kernel and the user–defined words added to the dictionary since boot–up.

**5. Discussion and Conclusions**

The system described above has been in class–room use since September 1984 and and has proved to be robust. The 16–bit model is adequate, even in a 32–bit environment, although beginners frequently forget to ensure that the code–field is properly aligned on a 32–bit boundary. A more serious fault with the implementation, is the cumbersome procedure needed to install a machine–code primitive as described in Section 3. The kernel source and the C–routine source have to be modified and both of them have to be re–processed. This task takes about thirty minutes, so that a better technique for interfacing Forth to C–coded primitives should be devised.

The department has acquired a Convergent Technology MegaFrame with two MC68010 application processors and a micro–VAX computer, both operating under Unix. It is planned to implement fig–Forth on these machines as implementations now exist for both the VAX–11 and MC68000 processors. Forth–83 is not being neglected, as it is the preferred version of Forth for all personal computers to be bought by the Department.

**6. References**

[1] Unix is the trade mark of AT&T Bell Laboratories.

[2] "TTL 68000–Forth, Edit Lev. 13/21–Aug–83," Christoph P. Kukulies, Heider Hof Weg, Aachen, 5100 West Germany, unpublished (1983).

[3] N. Solntseff, "An Abstract Machine for the Forth System", *Proc. 1982 Rochester Forth Conference*, I.A.F.R. Inc., Rochester, N.Y. 14611, pp. 149–155 (1982).

[4] Pixel Computer Inc., "PIXEL/UNIX Version 2.10, Pixel 80 or 100/AP", Wilmington, MA 01887 (1984).

[5] P.R. Blake and N. Solntseff, "A MC6800 fig–Forth Implemented under Unix," Technical Report, Department of Computer Science and Systems, McMaster University, (in preparation).

[6] A. Winfield, *The Complete Forth*, Sigma Technical Press, Wilmslow, Cheshire, U.K. (1983).

[7] A. Anderson and M. Tracy, *Mastering Forth*, Micromotion, Los Angeles, CA 90025 (1984).