# Adding a Third Stack to a Forth Engine

*Rick VanNorman*
*Phil Koopman*

*Harris Semiconductor*

## *Abstract*

Of the many possible uses for an extra stack in a Forth execution environment, a local variable stack is the most attractive. A local variable stack should be memory-resident, and be accessed via an offset to a frame pointer register. This arrangement provides excellent support for local variables in Forth as well as a clean interface to C and other programming languages.

## *Introduction*

Many uses have been proposed for a third stack in the Forth execution environment. Some uses include: local variables, loop control parameters, and separate stacks for each data type (*e.g.*, a string stack, a floating point stack).

The problem is that silicon real estate is expensive, so there must be a compelling reason to add an additional stack. Since Forth has long been successful with a two-stack execution model, it is likely that the impetus for adding an additional stack will come from requirements for supporting other languages. Only a stack for local variables is important and inexpensive enough to support with hardware on a Forth engine.

## *Local Variables*

Local variables are variables that are visible only within a small region of a Forth program, typically a single word. Various schemes for supporting local variables in Forth have been discussed for years (such as in **[HART85]**, **[LAQU84]**, and **[LYON85]**). The purposes of local variables are to make code more understandable and to reduce the number of stack manipulations. They require more or less random access to a group of values that are typically declared at the beginning of a word and discarded at the end of that same word.

Implementations can use several locations for local variables: statically allocated space dedicated to the Forth word, statically allocated global shared memory, the data stack, the return stack, and a separate local variable stack. All the schemes except a local variable stack have limitations or efficiency problems. The local variable stack has the problem that it requires adding another stack to the run-time environment.

## *A Solution*

C is an important market force for Forth engines. Any commercially successful stack computer must be able to run C well.

C programs use activation records for storing their local variables. An activation record is a stack that is allocated in blocks, or frames, and accessed using offsets to a pointer to the top of the stack. Activation records support random access to local variables, and recursive calls. Stack computers will support activation records in hardware to gain C performance.

The C activation record exactly matches the requirements of Forth local variables. If Forth programmers adopt the standard C activation record format for local variables on their system, they will accrue a number of benefits: local variables will be supported with random access, values from the data and return stacks can be flushed to the activation record prior to recursing to eliminate the possibility of stack overflows, and Forth programs will have ready access to parameters passed from C programs.

## *Implementations*

On general purpose hardware, local variables can be implemented simply by following the conventions for C parameter passing. On 680x0 systems, this typically involves using an address register as the frame pointer for the local variable stack. On 80x86 systems, this typically involves using the BP register (which may conflict with using this register as the return stack pointer or data stack pointer on many systems).

Many Forth engines already have some support for a frame pointer, since running C efficiently is becoming an important market issue. For example, the RTX 2000 has a relocatable user area pointer that is used to support C activation records.

## *An Example*

For an example of the implementation of the proposed local variable scheme on Forth hardware, consider the RTX2000. It possesses a USER-BASE register, which can be used as an external local variable stack. It allocates space in chunks of 32 words, and reading or writing any location of the user space requires two clock cycles. If we assume that local variables will be mapped into this space, and we assume a smart C compiler (which is beyond the scope of this paper), we can see the mechanism by which the RTX2000 can efficiently use its user space for local variables.

A simple implementation (adapted from **[LAQU84]**) of a word that benefits from the use of local variables is **QUADRATIC** which is the solution of the quadratic equation $Q = Ax^2 + Bx + C$. A typical Forth implementation of **QUADRATIC** is:

```
: QUADRATIC ( c a b x - q)
   ROT OVER DUP * * -ROT * + + ;
\ 18 clock cycles, RTX2000
```

which is as clear as mud.

If a smart implementation of local variables is assumed, which can compile the proper code, a better implementation is possible:

```
0 LOCAL X
1 LOCAL B
2 LOCAL A
3 LOCAL C
: QUADRATIC ( c a b x - q)
   [ 4 LOCALS ]
   A X X * * B X * + C + ;
\ 25 clocks, RTX2000
```

which compiles to (in an almost-assembler syntax) on the RTX2000:

```
LABEL QUADRATIC ( c a b x - q)
   0 U!  \ set up the local frame
   1 U!
   2 U!
   3 U!
   2 U@  \ A
   0 U@  \ X
   0 U@  \ X
   *
   *   \ AX^2
   1 U@  \ AX^2 B
   0 U@  \ AX^2 B X
   *   \ AX^2 BX
   +   \ AX^2+BX
   3 U@  \ AX^2+BX C
   +   \ AX^2+BX+C
   EXIT
```

Note that the calling routine is responsible for allocating the stack frame that the routine used. This is identical to the code generated by the C routine (also for the RTX2000):

```
int quad (c a b x)
  int c, a, b, x;
{
   return (a*x*x + b*x + c);
}
```

Programs as simple as this gain clarity, but not necessarily speed, from the use of an activation record; as the program becomes more complex with more than two or three items on the stack, stack thrashing to keep the correct operands on top outweighs the overhead of building and accessing an activation record. Another benefit of using an activation record is in the resulting clarity of the code written. The stack contains only the items of interest for each operation, not the entire state of the local variables known to the routine. The resulting code written for this model will be easier to understand, debug, and maintain.

## *Conclusions*

A memory-resident local variable stack for Forth can be implemented using hardware that is already provided on processors in support of the C programming language. This provides efficient support for local variables as well as a gateway to interfacing Forth programs to programs in C and other languages.

## *References*

[COLL85] Collins, L., "LOGO in Forth - A Role for Stack Frames and Local Variables," *1985 FORML Proceedings*, 59-70.

[HART85] Hart, J.R., and Perona, J., "Local Variables," *The Journal of Forth Application and Research*, 3-2 (1985), 159-62.

[LAQU84] LaQuey, R., "Local Variables," *1984 FORML Proceedings*, 307-316.

[LYON85] Lyons, G.B., "Stack Frames and Local Variables," *The Journal of Forth Application and Research*, 3-1 (1985), 43-52.

[STOD85] Stoddart, B., "Readable and Efficient Parameter Access Via Argument Records," *The Journal of Forth Application and Research*, 3-1 (1985), 61-82.

[VOLK84] Volk, W., "Named Local Variables in Forth," *1984 FORML Proceedings*, 307-316.