Updating the Forth Virtual Machine

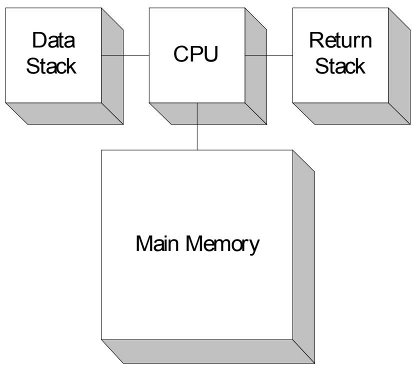
Stephen Pelc
MicroProcessor Engineering
133 Hill Lane
Southampton SO15 5AF
England
tel: +44 (0)23 8631 441
fax: +44 (0)23 8033 9691
net: sfp@mpeforth.com
web: www.mpeforth.com

Abstract

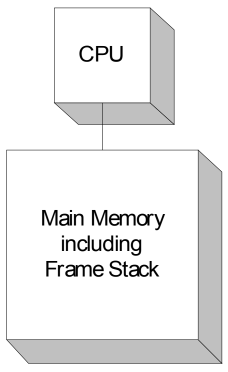
The canonical Forth Virtual machine has remained essentially the same since its inception. Modern silicon implementations and compiler techniques indicate that the VM as used in practice differs from this model. It is time to consider overhauling the canonical Forth Virtual Machine. In particular, the addition of address registers is considered.

Introduction

Classical or canonical Forth views the world as a CPU connected to main memory and two stacks which are not addressable, and are quite separate from main memory. C views the world as a CPU connected to memory, which includes a list of frames (usually a stack of frames) which **must** be in addressable memory.

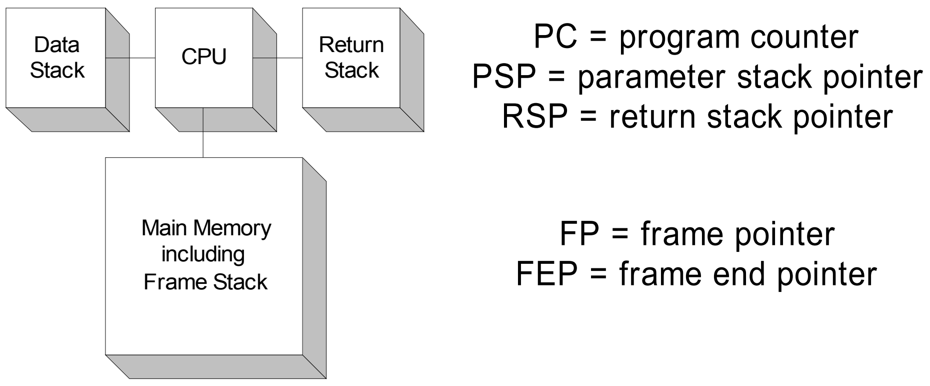


Forth VM



C VM

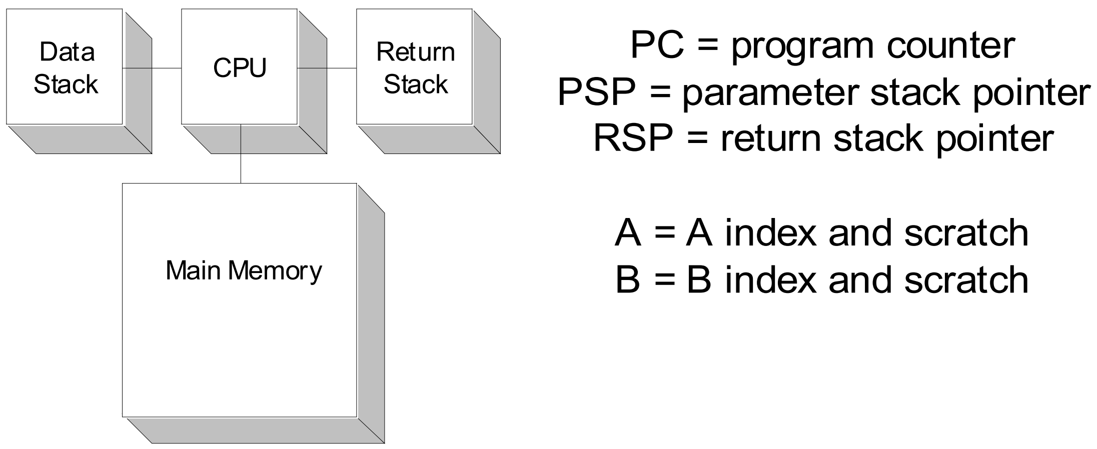
By adding the necessary registers for the frame stack to the canonical Forth machine, we arrive at the basic design of the SENDIT VM, which was discussed in various papers in the late 1990s. SENDIT (EP9152) was a project carried out under the European Union's ESPRIT research and development programme. SENDIT was based upon the results of a preceding project, PROCIC EP5497, and produced tools for the development of heterogeneous networks for use in embedded and real time applications.


Data Stack
CPU
Return Stack
Main Memory including Frame Stack
PC = program counter
PSP = parameter stack pointer
RSP = return stack pointer
FP = frame pointer
FEP = frame end pointer

SENDIT VM and registers

The SENDIT VM looks remarkably similar to other stack machine CPUs derived from a Forth architecture and designed to execute C efficiently.

Another branch of the Forth virtual machine has been called machineForth, and appears in software implementations such as ColorForth and various CPUs from iTV, Ultratechnology and IntellaSys, most lately in the SEAForth S24 multicore chips.



machineForth VM and registers

Other CPU core designs include MicroCore and designs from Bernd Paysan, Brad Eckert and Chris Bailey.

What distinguishes these cores is that they introduce data cells, registers and operations that are unsupported by the canonical Forth machine. In the description I have chosen not to include the TOS, NOS and TOR virtual registers. TOS and NOS are common across virtually all implementations as ALU inputs and outputs. TOR has wide variation in implementation for anything other than to hold a return address.

This paper explores the impact of these designs on how the Forth programming language could be changed.

Why update the Forth Virtual Machine?

The canonical Forth virtual machine is weak in several areas.

1) It does not execute C well, which is important for commercial exploitation of silicon stack machines.
2) It is weak for DSP operations, which restricts performance in embedded applications without changes to the VM or much increased compiler complexity,
3) Without index operations, it is cumbersome to deal with complex data structures whose base address is passed as an argument to a word.

Execution of C requires a frame pointer for access to local variables and buffers.

DSP operations often require three or four parameters to be manipulated regularly, e.g.

1) source address, destination address and length,
2) first source address, second source address, destination address and length.

Canonical Forth requires ugly source code to deal with these situations. Silicon implementations such as C18, FR32 and the Teesside University machines have provided index and scratch registers, whereas others have provide more access to the top of the return stack. Using the top of the return stack as a loop counter has been common for some time, e.g. the **`FOR ... NEXT`** loop structure.

The Forth community has long talked about TOS (top of data stack), NOS (next/second on data stack) and TOS (top or return stack). These are not quite enough for DSP operations an Chuck Moore's current silicon includes A and B registers which are used both as index registers and for scratch storage.

A new Forth Virtual Machine

I claim no particular novelty in this machine. It is a synthesis of practice that has been observed in several software and silicon machines over the years. What triggered this paper was seeing that Forth various compilers, e.g. Gary Bergstom's AFT (Another Forth Translator) have either implemented additional registers and facilities in their Forth VMs, or are seriously considering doing so.

If we look at what is common between these designs we find the following that can be treated as registers rather than just as ALU connections.

A	Register used as a scratch or index register, often with auto-increment and/or auto-decrement addressing.
B	Register used as a scratch or index register, often with auto-increment and/or auto-decrement addressing.
LP	Local frame pointer with base+literal indexed addressing.
UP	User area pointer with base+literal indexed addressing for thread-local storage.

Table 1: Additional Forth VM registers

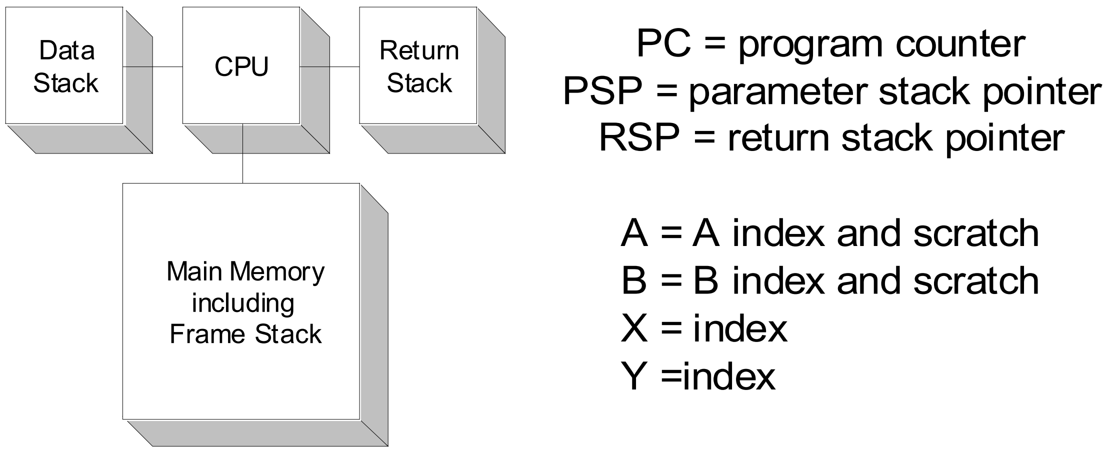
Inspecting various Forth implementations and source code, we can make various observations:

1) Use of the A and B registers considerably reduces the need for local variables.
2) Use of the A and B registers can considerably reduce stack manipulation in both source and compiled code.
3) Although UP can be implemented as a variable, most Forth systems, especially embedded systems, implement it using a CPU register.
4) What distinguishes the A/B pair and the LP/UP pair is that A/B implement auto-increment addressing, and occasionally auto-decrement addressing. The LP/UP pair implement base + offset addressing.
5) The use of the scratch registers improves source code density (level of abstraction) and reduces stack shuffling at basic block boundaries and avoids complexity in code generators.

In order to avoid mandating use of these registers, we can simply rename them in terms of how they are used:

A	Register used as a scratch or index register, often with auto-increment and/or auto-decrement addressing.
B	Register used as a scratch or index register, often with auto-increment and/or auto-decrement addressing.
X	Memory pointer with base+literal indexed addressing.
Y	Memory pointer with base+literal indexed addressing.

Table 2: Additional registers in the new VM



A possible new Forth VM and registers

Wordsets

A and B registers

This a fully featured wordset. Some systems only provide auto-increment/decrement on the A register. On some systems, the B register cannot be read. The A and B registers provide the source and destination address pointers used for block, string and DSP operations as well as providing scratch storage.

`>A` `\ x --`
Writes to the A register.
`>B` `\ x --`
Writes to the B register.
`A>` `\ -- x`
Reads the A register.
`B>` `\ -- x`
Reads the B register.
`A@` `\ -- x`
Read the memory pointed to by the A register.
`A!` `\ x --`
Write the memory pointed to by the A register
`B@` `\ -- x`
Read the memory pointed to by the B register.
`B!` `\ x --`
Write the memory pointed to by the B register
`A@+` `\ -- x`
Read memory pointed to by A, increment A by one cell. A post-incremented read.
`B@+` `\ -- x`
Read memory pointed to by B, increment B by one cell. A post-incremented read.
`A@-` `\ -- x`
Read memory pointed to by A, decrement A by one cell. A post-decremented read.
`B@-` `\ -- x`
Read memory pointed to by B, decrement B by one cell. A post-decremented read.
`A!+` `\ x --`
Write to the memory pointed to by A, and update A.
`B!+` `\ x --`
Write to the memory pointed to by B, and update B.

X and Y registers.

The X and Y registers provide indexed addressing. In Forth they can be used to implement the **`USER`** area and local frame pointers.

`>X` `\ x --`
Writes to the X register.
`>Y` `\ x --`
Writes to the Y register.
`X>` `\ -- x`
Reads the X register.
`Y>` `\ -- x`
Reads the Y register.
`nX@` `\ n -- x`
Read the memory pointed to by the X register plus n (literal) address units.

nX! `\ x --`
Write the memory pointed to by the X register plus n (literal) address units.
nY@ `\ -- x`
Read the memory pointed to by the Y register plus n (literal) address units.
nY! `\ x --`
Write the memory pointed to by the Y register plus n (literal) address units.

Biquad filter example

My thanks go to Gary Bergstrom for permission to publish this code.

```
: *.       \ fr1 fr2 -- fr3
\ Fractional multiply.
  +1. */  ;
: 1STEP+   \ sum -- sum'
\ Perform a multply/accumulate step, incrementing both
\ pointers.
  B@+ A@+ *. +  ;
: 1STEP-   \ sum -- sum'
\ Perform a multply/accumulate step, incrementing the
\ coefficient pointer and decrementing the data pointer.
  B@+ A@- *. +  ;
: SHIFT2   \ fr --
\ The last step of the filter. The current data item
\ is shifted into the next data slot and replaced by fr.
  A@ SWAP A!+ A!+ ;
: (BIQUAD) \ frx -- fry
\ The core of the biquad filter operation.
  DUP >R
  B@+ *.              ( initial sum = B0*input )
  1STEP+ 1STEP-  R> SHIFT2
  1STEP+ 1STEP-  ;
: BIQUAD   \ fx addr-filt addr-coef -- fry
\ A single order biquad filter.
  >B >A  (BIQUAD) DUP SHIFT2  ;
: 2xBIQUAD \ fx addr-filt addr-coef -- fry
\ A second order biquad filter.
  >B >A (BIQUAD) (BIQUAD) DUP SHIFT2  ;
```

References and further reading

[1] SENDIT token specification, ISBN 0-9525310-1-1, MicroProcessor Engineering, 133 Hill Lane, Southampton, England

[2] Europay Open Terminal Architecture
Volume 1 - Token Specification
Volume 2 - Forth language binding
Volume 3 - C language binding
MasterCard International, 198A Chaussée de Tervuren, 1410 Waterloo, Belgium

[3] The evolution of SENDIT into EPIC, Stephen Pelc, Rochester Forth Conference 1996

[4] The SENDIT project: a Forth in sheep's clothing, Jon Lee, Rochester Forth Conference 1996

[5] SENDIT tool architecture, ISBN 0-9525310-2-X, MicroProcessor Engineering, 133 Hill Lane, Southampton, England. Out of print.

[6] MicroCore:
http://www.microcore.org

[7] Stack Computers: the new wave, Philip J. Koopman, Jr.,
http://www.ece.cmu.edu/~koopman/stack_computers/index.html

[8] IntellaSys SEAforth-24,
http://www.intellasys.net

[9] C18, Chuck Moore,
http://www.complang.tuwien.ac.at/anton/euroforth/ef01/

[10] B16, Bernd Paysan,
http://www.jwdt.com/~paysan/b16.html

[11] AFT, Gary Bergstrom
private communication