A Processor as Hardware Version of the Forth Virtual Machine

Willi Stricker
Springe, Germany,
September, 12, 2011

The structure of the Forth system

in contrast to other programming languages the Forth System consists of several components:

- Forth virtual machine
- programming language
- operating system

The Forth virtual machine
In general the Forth virtual machine is based on a microprocessor of any kind and created by software. It is, at least theoretically, identical for every microprocessor and has a unique interface between hardware

The real Forth processor
It is obvious to combine the hardware part of the Forth virtual machine with its sofware part to a real Forth machine that is made of hardware exclusively. This is now called the real Forth processor and is possibly made for instance by means of a modern programmable hardware (programmable logic like FPGAs).

To evaluate its properties, ist is necessay to find the differences between the Forth virtual machine and a „general“ microprocessor of common construction. Mainly there are two features:

1. There is a parameter stack instead of the customary registers
2. Every instruction is just an address.

The Forth virtual machine and the Forth system have two kinds of instructions:
1. Primitives
They are written in assembler code of the used processor. They correspond to the machine code instructions (assembler instructions) of a general microprocessor.
2. High level instructions
Instructions that ar written in Forth (instructions that are combined of primitives). They correspond with subroutines of general microprocessors.

As mentioned bevore, both kinds of instructions in a Forth system are represented by addresses and are accesed by them.

The Forth virtual machine contains an „inner interpreter“ mostly named „NEXT-routine“. It is a „program switch“ for both kinds of instructions. For distinguishing them the instructions need an instruction prefix that is a (sub)program address. For primitives that is the address of their own assembler program, for high level instructions it is the address of a small subroutine (mostly called „DOCOL-routine“), pushing the return address to the return stack and branching to the subroutine (replacing a call instruction). High level instructions are terminated by a return instruction (;S = SEMIS-routine), that returns to the calling instruction by popping the address from the return stack and jumping to it.

The coclusion is: **There is no explicit call instruction in a Forth system.**

The Forth virtual machine makes no statements about hardware properties. That is especially true for the interrupt system. Generally the hardware of the microprocessor is used. But by the way that is true for all common programming languages.

Demands for a real Forth processor
It should have the same (software) features like the Forth virtual machine and additional hardware features of a micro processor.

The construction of a real Forth processor

Afterwords a model is shown that meets the above conditions as much as possible.
It got the name **STRIP (STack Related Instructions Processor)**

it contains the following hardware equipment:

- a separate parameter stack and a separate return stack each
- an interrupt interface
- the means to connect external or internal memory
- the means to connect internal or external input and output elements

it uses only three pointer registers:

1. parameter stack pointer: SP
2. return stack pointer: RP
3. instruction pointer: IP

There are no additional pointers or registers (especially no flag register!)

Minimum instruction set:
In Forth systems generally a set of primitives are available programmed in assembler. For the real Forth processor they have to be constructed in hardware. To minimize the hardware a “minimum instruction set“ is provided chosen due to the following critera:
It contains only instructions that are mandatory for the design of a complete Forth system and additional instructions that are mandatory for hardware control.

System instructions (used by the compiler only)

;S	(-)	return = pop the address from the return stack and store it to IP
LIT	(- data)	load immediate data on stack
BRANCH	(-)	branch to the following address
?BRANCH	(data -)	branch to the following address if data equals zero, else continue

Indirect instruction and subprogram call

EXECUTE	(address -) execute instruction (address on top of stack)

Access to the parameter and the return stack pointer

RP@	(- RP) get RP
RP!	(RP -) store RP
SP@	(- SP) get SP
SP!	(SP -) store SP

Return-Stack manipulation:

R>	(- data) pop data from return stack
>R	(data -) push data to return stack

Parameter-Stack manipulation (random access to the parameter stack):

DROP	(data -) drop data from stack
PICK	(position - data) load data from relative stack position
-PICK	(data position -) store data on relative stack position

Memory access

@	(address -> data) fetch data from memory address
!	(data address ->) store data on memory address

Logic functions

INVERT	(data - result) bitwise NOT
AND	(data1 data2 - result) bitwise AND
OR	(data1 data2 - result) bitwise OR

arithmetic functions

+C	(data1 data2 - result carry) add data1 to data2 with sum and carry (lsb)
U2/C	(data - carry result) shift right one bit, with result and carry (msb)

Special byte instructions
CSWAP (byte1|byte2 - byte2|byte1) swap bytes in data
C@ (address - 0|byte) fetch byte from address (upper byte = 0)
C! (byte address -) store byte to address (only lower byte, upper byte is discarded)

Processor control instructions (interrupt instructions)
DISINT (-) disable Interrupts
ENINT (-) enable Interrupts

These 26 instructions are sufficient for the construction of a complete Forth system (remark: some of these instructions are not defined or usual in Forth).

The real Forth processor has two particularities against the Forth virtual machine.

1. No instruction prefix
as discribed earlier, the Forth virtual machine needs an instruction prefix to distinguish primitives and high level instructions by the NEXT routine. This is omitted in the real Forth processor. So memory space and access time is saved. Now the type of insruction has to be determined by the program address. While the primitives don't have real addresses pseudo addresses are created instead in a reserved address space.

2. Return bit
Every code address is an even one (two bytes of code). So the right most bit (least significant bit) is always zero! This bit is free for another information and used as „return bit“. The return bit causes a return after the addressed instruction has been executed, independently weather the instruction is a primitive or a high level one. In a subroutine the last instruction gets a return bit. The explicit return instruction is omitted.

While programming there are situations, that need an explicit return instructions. So in the previously defined instruction set a new insruction is inserted (NOP = no operation) and the return instruction is replaced by a NOP with a return bit:

NOP (-) No Operation

The following special cases need a return instruction with address:
Trivial case: A subprogram must have at least one instruction, even if it does nothing. The only instruction is the return.
Structures (branch instructions): A branch always needs a valid address to branch to, that mostly is the address of the following instruction. But on the end of a subprogram there is no instruction left, so it needs a NOP (with a return bit).

The STRIP Forth processor in reality

First objective of its construction obviously is high speed. So as much as possible has to be executed concurrently. Then the time of execution is resricted only by accessing external memory (access time). This limits the minimum of time. Therefore as few as possible accesses to the memory (bus) should happen.
The stacks are separated physically from the main memory. They are accessable by stack instructions only. As a result the processor has access to both stacks and main memory concurrently.

The STRIP kernel contains the three pointers, their control and the whole instruction set. It uses a clock-signal and has interfaces for the parameter stack, the return stack, an interrupt interface, and data and address busses for memory and peripheral elements (memory mapped I/O).
The kernel is extended to a working processor by adding a clock element, parameter and return stacks and an reset/interrupt controller.
A complete working STRIP Forth system needs additional memory (RAM and/or ROM) for data and programs, as well as a program and debug interface. Furthermore it is possible to add input/output elements.

Timing
The Subroutine call and most of the primitives need only one bus access and with it only one clock cycle! Two bus accesses and clock cycles are used only for instructions

- that need an additional parameter: **LIT, BRANCH, ?BRANCH,**
- that access memory: **@, !, C@, C!,**
- that need a second cycle with a return bit: **R>** with Return bit**, RP!** with Return bit.

Notes:
instructions with operands need two memory spaces, instructions with memory access only one. Instructions with a return bit don't need an additional bus access and no additional time. Exceptions only the instructions RP! and R>, they need a second cycle for the return (without bus access).

Pseudo addresses for primitives
In the Forth virtual machine the primitive instruction are assembler code with memory addresses. For the STRIP Forth processor pseudo addresses are defined. These addresses (numbers) address primitives and are not usable for high level instructions. But they can be used for memory. The pseudo addresses are placed into an area outside the program memory. It is a good idea to put them on the start of the memory map (starting with addrerss zero).

With the minimum instruction set 26 addresses are used. In a 16 bit system 52 bytes are necessary. In practice 64 (2^6) bytes are reserved for pseudo addresses.

Restart and interrupts
For the restart memory space is reserved for the restart address. For the interrupts there is also one space each for its address that contains the start of its corresponding interrupt service routine.

The restart is an primitive, activated by hardware. It needs one clock cycle (address fetch).
An interrupt is activated by hardware asynchronously. It is checked in S0 by the kernel and executed after the concurrently fetched instruction is executed. The interrupt doesn't need any overhead time beside its own program.

Final conclusion and construction

The STRIP Forth processor is firstly programmed into an FPGA (Actel eval kit with APA 075 with 3075 Tyles and 3 kb RAM) that is sufficient for a 16-bit system for experimental proving of the processor.