Our 4e4th05a vs CamelForth

Why did we design the 4e4th based on CamelForth?

We needed a Forth that was adapted well to our experiments using the TI MSP430G2553 Launchpad. You can start with a Breadboard compatible DIL version and use smaller variants where needed. This Forth should be very error-tolerant and practical when working together with the 4e4th-IDE programming environment to be found at http://www.4e4th-ide.org/. At that time in 2012, CamelForth http://www.camelforth.com/news.php was the best starting point we had seen. But the constant changes that IAR introduced for "Kickstarter" IDE was not helpful, and unfortunately CF was written in it at the time. This commercial software became more and more of a hindrance adding complexity that was not needed here.

But now there is Mike Kohn's naken assembler https://www.mikekohn.net/micro/naken_asm.php. And Brad Rodriguez has ported and improved his CamelForth using this tool. CF now offers tools like MARKER, SAVE and RESTORE, as well as good ways to restore an intact Forth kernel - even if garbage has been programmed. This is achieved interactively via the terminal, using SRCUB or via the hardware of the launchpad - the two buttons S1 and S2. So, hardly any changes were needed to get our good 4e4th look & feel back in this new version.

Developing Applications.

CamelForth/MSP430 has the ability to save and autostart user-defined application programs. 4e4th05a works exactly in the same way. That was not been changed.

Read the appropriate chapter in **msp430development.pdf** to understand how it works.

This also applies to the following chapters.

SCRUB and RESTORE

n 4e4th, there was WIPE instead of SCRUB. Both do work identically. In 4e4th05a, **both** have been included for backward compatibility.

In 4e4th, WARM was used as a starting option in addition to COLD. This notation was still from the old AIM65 Forth times. CamelForth can perform a RESTORE after COLD, which equates to a Warm Start if a SAVE had been done. For these commands the following applies:

They are provided for Development and Debugging purposes. There should never be a need in your application program to call RESTORE or COLD or SCRUB.

Autostart Bypass

4e4th had this, and CamelForth implemented it too. So it works in both of them in the same way.

Hold down S2 button and press S1 reset button then.

Summary of behaviors

See: msp430development.pdf, page 4.

4e4th and CF have identical behaviors.

Creating an autostartable Application

Same as in CF, see the PDF:

Application programs must reside entirely in ROM! The word which starts your application must be a colon definition, and must be the last word you compile.

MARKER

Same as CF, see the PDF:

If you use MARKER, normally you'll just use one, so this isn't a major problem. After you've finished debugging, and are ready to compile your completed application, you should remove the MARKER from your source file.

Set, clear and test bits in a cell.[1]

CamelForth already supports this nice MSP430 instructions.

Clear, set or test a bit in a data **word**.

CLRB	(mask adr – –)
SETB	(mask adr – –)
TSTB	(mask adr – –)

Clear, set or test a bit in a data **byte**.

CCLRB	(mask adr – –)
CSETB	(mask adr – –)
CTSTB	(mask adr – –)

So, what is different in this new 4e4th?

No case sensitivity while typing

CF is case sensitive, 4e4th is not. In the older 4e4th this was switchable, now it is an integral part of 4e4th05a.

The OK-Prompt

After interpreting a line of code, Forth prompts with OK if no error occurred. CF sends a simple OK. 4e4th displays the current number base as well. The format is: $hh

You will see one of these three options:

`$0Aok`	decimal number base.
`$10ok`	hexadecimal number base.
`$02ok`	binary number base.

You can choose either of them by using BASE and by typing `<value> BASE !`

The prompt pictures the stack by printing a dot for each value on the stack. An empty stack has none.[2]

`$0Aok`	Empty stack.
`$0Aok..`	Two values on stack.
`$10ok...`	Three values on stack.

… and so on.

Stack underflow (SUF) detection

In 4e4th, ?STACK aborts compilation if a stack underflow occurs, and prints SUF, otherwise no other effect.

---

1 In old 4e4th we named them CLR SET GET. Now we use the CF notation, so this is *not different* any more.

2 Thanks to noForth.

Additional features and words

See the Forth word glossary to learn the syntax and how to use these words. The expression in brackets following a word is a stack comment. In Forth, as you may know already, words put or get values from the data stack. The left part shows values infoled on the stack already, the right part after execution.

(– –) This word will not have a stack effect.

(n – –) Word will consume the number n from stack.

(– – n) Word will leave the number n on the stack.

(n – – m) Word will consume number n from stack, and leave number m.

… and so on. The same applies to multiple numbers used within a word.

WORDS

It has a stop&go feature included: Press the space-bar to stop listing or continue; any other key will abort the display listing.

WORDS (– –)

\

The backslash character is provided to suspend compilation until the end of a line. Used for writing comments in the same line.

\ (– –)

WIPE

The same function as SCRUB – kept for compatibility reasons: Make a clean user flash.

WIPE (– –)

2CONSTANT GREEN RED S2 S2?

CF and 4e4th use the definition of (mask adr --) notation to address a single bit in a memory word or byte, or in a port bit. So, 2CONSTANT is provided to save mask and addr as one word. GREEN would be such a Word: it puts mask and address of the port pin corresponding to the green LED onto the stack—the pin on the Launchpad of course. RED does the same for the red LED. And S2 does it for the S2-button, so you can read it's state. S2? already does the complete job by providing a flag on the stack indicating the current status of S2.

These words are provided to enable quick first steps in Forth using the TI Launchpad hardware.

2CONSTANT	(n – –) <name>
GREEN	(– – mask adr)
RED	(– – mask adr)
S2	(– – mask adr)
S2?	(– – flag)

CTOGB

To toggle a bit in a byte. 4e4th has it. An easy way to let your LED's blink.

CTOGB (mask adr – –)

P1OUT P1IN

To control the output and input register address of Port1.

P1OUT (– – adr)

P1IN (– – adr)

1MS

A delay of about 1 millisecond. This is done via nested loops, not by a calibratable counter.
So, do not expect too much precision.
1MS (– –)

MS

Delay of several milliseconds. The number is on the stack and interpreted using the current number base. E.g. `1000 MS` is a second if the number base is decimal – 4096 if current base is hex.

MS (n – –)

So, this is it for the moment . Have fun! Version 2018_05_12
Michael

Any errors or typos please send to mik.kalus@gmail.com