# How many stacks?!

Forth actually has two stacks. The data stack is where we are storing the parameters that our subroutines act on. Any values being returned from a subroutine are left on the data stack for other code to process.

There is also a return stack, which stores addresses to be used to return to the relevant place in the code when a subroutine has finished or during a loop. Some Forth programs use this as an 'extra hand' when manipulating the data stack.

We can see the pointer to the top of the return stack by using r@. There is also a pointer to the data stack sp@ but the repl.it version of Forth does not support it, nor the sp0 word which gives us a pointer to the bottom of the data stack. A further developed version of this session might show how these pointers move and give an even deeper understanding of how the stack works. Use the free [GForth](https://www.gnu.org/software/gforth/) implementation if you want to explore this further.

The underlying operating system is also making use of a hardware stack (so called because the CPU itself pushes PC onto the stack during subroutine calls and uses a stack pointer register to manage the stack). So in a Forth system running on a typical desktop OS, there will be at least 3 stacks in use. A Level students don't need to know this detail for their exams but it is worth us knowing that when we are referring to 'the stack' in the syllabus, we are talking about the system stack, or an abstract data type that is used in high level code.

# Links to Assembly Language/Hardware

<https://peterhigginson.co.uk/ARMlite/>

The above new version of LMC appeared fortuitously just before I was due to deliver this session at CAS London February 2020. It allows a further insight into the stack at the hardware level as if you switch to code page FFF, and single step through the instructions below you can actually see the hardware stack pointer register, the stack itself growing down from memory location FFF, and the effect on the memory locations as push instructions are executed.

MOV R0, #1

PUSH {R0}

MOV R0, #2

PUSH {R0}

POP {R0}

POP {R0}

It's much easier for students to run the above than explain it, and the same goes for Interrupt Service Routines (ISRs) and the need for the programmer to use the stack during the execution of these routines.

See my demo code below for an example. It outputs the numbers 1 to 12, 12 times and then starts again. If you click the lightning bolt symbol a hardware interrupt will occur. Nothing appears to happen, but if you stop or pause the simulator you will see the value at testLocation (ffffffff) has been overwritten with 1.

The code as provided saves the value of R0 to the stack on entry, and then pops it back off before returning to wherever the interrupt happened, if it did not save the value of R0, the count would go back down to one as R0 is being used in the main code to hold the count in the outer loop.

Ask the students to remove the lines that pop and push from the provided code. Get them to single step the code, by pressing the https://peterhigginson.co.uk/ARMlite/images/Step.png symbol. Let the loop run at least twelve times so that 2 is being repeatedly output to the screen. Hit the lightning symbol, and then carry on single stepping. You should see the code stepping through the ISR, and then returning to the main code. However the count should now have restarted at 1, because we corrupted the R0 register inside the ISR. This should illustrate why it is necessary to preserve register values on the stack if those same registers are going to be used during an interrupt.

MOV R0,#isr1

STR R0,.PinISR

MOV R0,#1

STR R0,.PinMask

STR R0,.InterruptRegister

start: mov r0,#0

outer: add r0,r0,#1

mov r1,#0

inner: add r1,r1,#1

cmp r1,#12

str r0,.WriteUnsignedNum

bne inner

cmp r0,#12

bne outer

b start

isr1: push {r0}

mov r0,#1

str r0,testLocation

pop {r0}

rfe

testLocation: .WORD 0xffffffff

# Further Forth

If this session has piqued your interest, you might want to explore the wonderful (though now out of print and rare) book by Leo Brodie called Starting Forth which has been made available here as a series of web pages:

<http://home.iae.nl/users/mhx/sf.html>

If you've really got time on your hands and want a broader picture of how Forth fits in to wider software development good practice, read Leo Brodie's Thinking Forth, which is available here:

<https://sourceforge.net/projects/thinking-forth/>

# Quick Reference to Stacks in Educational Computing Books

I really like the two A Level text books produced by Kevin Bond for his company Educational Computing. Stack related fun can be found on:

Unit 1

P120 The role of the stack in recursion, stack overflow

P140 -141 OO example of implementation of a stack

P231 The stack data type

P232 The uses of stacks

P120, 233 The stack frame

P318 - 320 RPN and stack machines

Unit 2

P228 The stack machine

P282 The stack pointer register