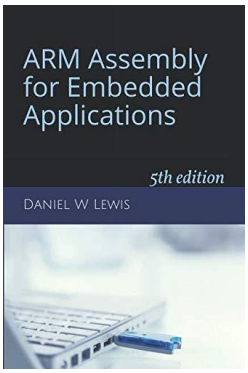

ARM Instructions Worksheet #1

Function Call and Return

And their effect on registers PC, LR, and SP.

Prerequisite Reading: Chapter 3: Sections 3.1 and 3.2
Revised: March 26, 2020

Objectives: To become acquainted with the web-based simulator ("CPUlator") and to use it to better understand how the …

1. Program Counter (PC) is used to fetch an instruction,
2. Branch and Link (BL) instruction is used to call a function,
3. Branch Indirect (BX) instruction is used to return from a function,
4. Link Register (LR) is used to hold the return address, and
5. PUSH and POP instructions use the Stack Pointer (SP) to preserve and restore register content.

To do offline: Answer the questions that follow the listing below. (Numbers at far left are memory addresses.)

```
                .syntax      unified
                .global      _start
00000000  _stack_end:
                .skip  100          // Reserve memory for stack
00000064  _tos:
00000064  _start:

00000064        LDR    SP,=_tos     // *** EXECUTION STARTS HERE ***

00000068        BL     f1           // Simple function call
0000006C        BL     f2           // Nested function call
00000070        BL     f3           // Optimized nested function
00000074        B      done         // End of demo

00000078  f1:   BX     LR           // Simply returns

0000007c  f2:   PUSH   {LR}         // Preserve LR
00000080        BL     f1           // Call f1 (changes LR)
00000084        POP    {LR}         // Restore LR
00000088        BX     LR           // Return (Copies LR into PC)

0000008C  f3:   PUSH   {LR}         // Preserve LR
00000090        BL     f1           // Call f1 (Changes LR)
00000094        POP    {PC}         // Return

00000098  done: B      done         // infinite loop

                .end
```

What is left in SP after executing the LDR instruction at 00000064_{16}? **the address of _tos or &_tos or 00000064 (base 16)**

What is left in PC after executing the LDR instruction at 00000064_{16}? **the address of the next step to be done or 00000068 (base 16)**

What instruction is at the address that's now in the PC? *(Include any referenced label)* **BL f1 // Simple function call**

What address is left in register PC after executing the `BL f1` instruction? **00000078 (base 16)**

What instruction is at the address that's now in the PC? *(Include any referenced label)* **BX LR // Simply returns**

What address is left in register LR after executing the `BL f1` instruction? **0000006C (base 16)**

What instruction is at the address that's now in the LR? *(Include any referenced label)* **BL f2 // Nested function call**

What value is in register PC after executing the `BX LR` instruction at 00000078_{16}? **0000006C (base 16)**

What instruction is at the address that's now in the PC? *(Include any referenced label)* **BL f2 // Nested function call**

Getting ready: Now use the simulator to collect the following information and compare to your earlier answers.

1. Click here to open a browser for the ARM instruction simulator with pre-loaded code.
2. Press Ctrl-E to open the "Editor" window and notice the LDR pseudo-instruction.
3. Press Ctrl-D to replace the editor by the "Disassembly" window. Notice how the LDR pseudo-instruction has been replaced by a ***real*** LDR instruction that loads SP from a word in memory whose content is the address of label "`_tos`" (top of stack).

Step 1: Executing the first instruction

The CPU registers are shown in the "Registers" window. Note that the PC value is 00000064_{16}. This is the starting address of the program. At that address is the LDR instruction that initializes the stack pointer (SP), highlighted in yellow to indicate that it is the next instruction to be executed. Press F2 **once** on the to execute that LDR instruction.

What is left in SP after executing the LDR instruction at 00000064_{16}? **00000064 (base 16)**

What is left in PC after executing the LDR instruction at 00000064_{16}? **00000068 (base 16)**

What instruction is at the address that's now in the PC? *(Include any referenced label)* **bl 0x78 (0x78: f1)**

Step 2: Call function f1

The PC should contain the address of the instruction, "`BL f1`". Press F2 **once** to execute the instruction.

What address is left in register PC after executing the `BL f1` instruction? **00000078 (base 16)**

What instruction is at the address that's now in the PC? *(Include any referenced label)* **bx lr**

What address is left in register LR after executing the `BL f1` instruction? **0000006c (base 16)**

What instruction is at the address that's now in the LR? *(Include any referenced label)* **bl 0x7c (0x7c: f2)**

Step 3: Return from function f1

The PC should contain the address of the instruction, "`BX LR`". Press F2 **once** to execute the instruction.

What is in register PC after executing the `BX LR` instruction at 00000078_{16}? **0000006c (base 16)**

What instruction is at the address that's now in the PC? *(Include any referenced label)* **bl 0x7c (0x7c: f2)**

Step 4: Continue exploring

Continue pressing F2 to step through the program, noting changes to registers `PC`, `LR` and `SP` at each step. Function `f2` contains a call to function `f1` that overwrites the return address of `f2` in `LR`. In order for `f2` to return properly, we use a `PUSH {LR}` at the entry of `f2` to copy the return address onto the stack and then restore it with a `POP {LR}` before the return. Function `f3` does the same, but eliminates the `BX LR` by popping directly into the PC.