# The ARM processor (Thumb-2), part 17: Prologues and epilogues

devblogs.microsoft.com/oldnewthing/20210622-00

June 22, 2021

Raymond Chen

The calling convention and ABI for ARM on Windows dictates a lot of the structure of function prologues and epilogues.

Here's a typical function prologue:

```
push    {r4-r7,r11,lr}      ; save a bunch of registers
add     r11, sp, #0x10      ; link into frame pointer chain
sub     sp, sp, #0x20       ; allocate space for locals
                            ; and outbound stack parameters
```

This is probably easier to explain with pictures.

On entry, the stack looks like this:

| | |
|---|---|
| return address | |
| previous *r11* | ← *r11* (frame chain) |
| ⋮ | |
| stack param | ← *sp* |

On entry to the function, *lr* contains the return address. After pushing the *r4* through *r7*, *r11*, and *lr* registers, we have

| | |
|---|---|
| return address | |
| previous *r11* | ← *r11* (frame chain) |

| |
|---|
| ⋮ |
| stack param |
| return address |
| previous *r11* |
| previous *r7* |
| previous *r6* |
| previous *r5* |
| previous *r4* ← *sp* |

The incoming *lr* is saved on the stack, so we know where to return to when we're done. The incoming *r11* is the head of the linked list of stack frames, and we push it onto the stack so we can create a new node on the linked list. And we also push four saved registers so that they are available for us to use in the function.

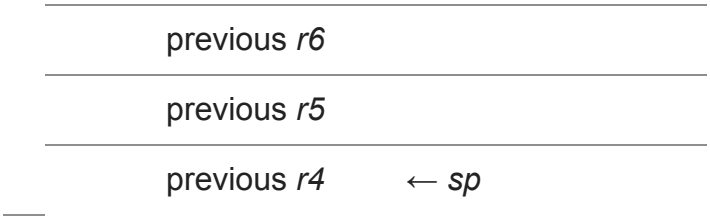
It is not a coincidence that the convention is to use *r11* as the frame pointer. This puts it on the stack right next to the *lr* register, so that the return address is right next to the frame pointer.[1]

The next instruction calculates *r11* as *sp* + `0x10`, which makes it point to where we saved *r11* onto the stack. This links a new node onto the stack frame chain.

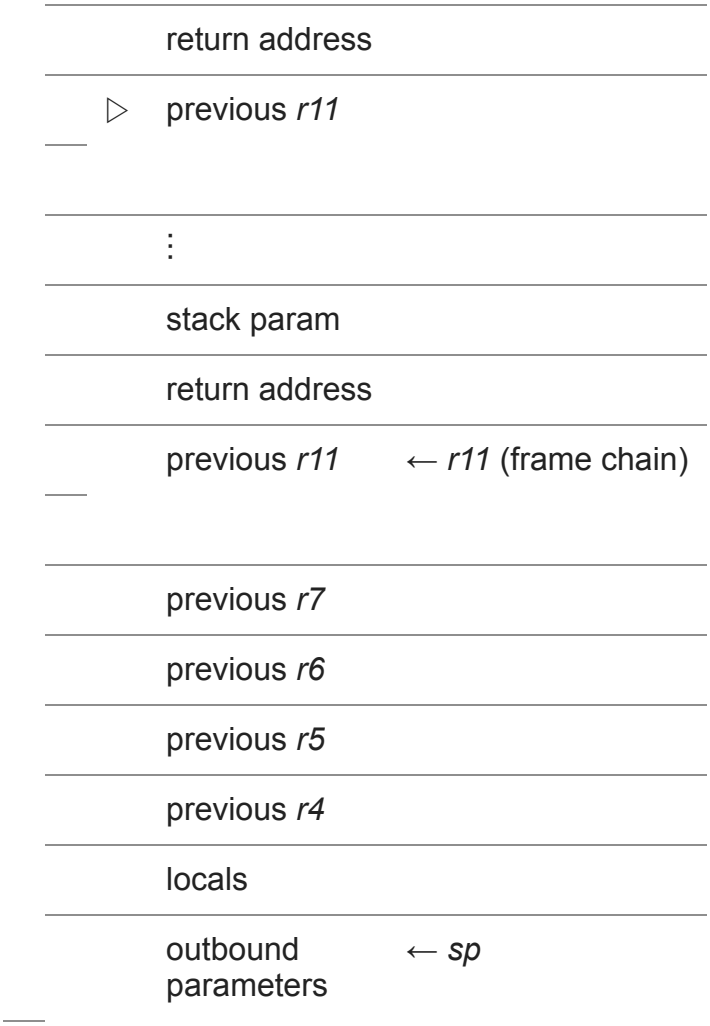
| | |
|---|---|
| | return address |
| ▷ | previous *r11* |

| |
|---|
| ⋮ |
| stack param |
| return address |
| previous *r11* ← *r11* (frame chain) |

| |
|---|
| previous *r7* |




And the last step in the prologue is allocating additional space for local variables and outbound parameters.




Windows does not require that the *r11* register be the head of a linked list of stack frames,[2] but all Windows system components are compiled with frame pointers enabled: It makes debugging a lot easier (since the `k` command always produces a stack trace), and it permits automated stack tracing, such as those created by `xperf`. In the stack frame chain, the return address is stored immediately adjacent to the *r11* pointer.

To return from the function, we run things in reverse:

```
add     sp, sp, #0x20       ; free locals and outbound stack parameters
pop     {r4-r7,r11,pc}      ; restore registers and return
```

The `pop` instruction is magic.

The obvious part of the `pop` instruction is restoring registers *r4* through *r7*.

The less obvious part is that we pop the original *r11* back into *r11*, which has the effect of deleting the frame from the linked list of stack frames.

The totally magic part is that we pop the return address (which was originally passed in *lr*) directly into the *pc* register. Writing to the *pc* register acts like a jump instruction, so this jumps to the return address after the work of this instruction is complete.[3]

The last thing the `pop` instruction does is update the stack pointer, which puts it back at the location it had when control originally entered the function. And then execution resumes at the return address.

The standard prologue looks like this:

```
push    {...,r11,lr}        ; save registers, frame pointer, return address
add     r11, sp, #nn        ; re-establish frame chain
                            ; can be "mov r11, sp" if only r11 and lr were pushed
vpush   {d8,...}            ; save floating point registers
sub     sp, sp, #nnn        ; create local frame
```

I call this the standard prologue because the function unwind metadata is optimized for prologues that take this form.

Next time, we'll look at some tweaks and optimizations to this general pattern.

[1] Now, there are two other registers in between *r11* and *lr*: We have the intraprocedure call scratch register *r12*, and we have the stack pointer *sp* (also known as *r13*). Fortunately, we can avoid having to push either of these two registers. The intraprocedure call scratch register is a volatile register that is not expected to be preserved, and the stack pointer is preserved either by keeping track of its value through the function (subtracting a frame on entry and adding it back on exit), or recovering it from the frame pointer. You aren't ever tempted to push the stack pointer because you cannot reliably pop it back anyway.

[2] The documentation is a bit unclear on this. In the discussion of the integer registers, it says

> Windows uses r11 for fast-walking of the stack frame. For more information, see the Stack Walking section. Because of this requirement, r11 must point to the topmost link in the chain at all times. Do not use r11 for general purposes—your code will not generate correct stack walks during analysis.

The use of the words *requirement*, *must* and *do not* imply that using *r11* as the frame pointer is mandatory.

But then when you get to the Stack Walking section, it says

> Generally, the r11 register points to the next link in the chain, which is an {r11, lr} pair that specifies the pointer to the previous frame on the stack and the return address. We recommend that your code also enable frame pointers for improved profiling and tracing.

This time, the use of the words *generally* and *recommend* imply that using *r11* as the frame pointer is merely a suggestion, albeit a strong suggestion.

I'm not sure who is right, but I'm going to assume that the use of *r11* as a frame pointer is *strongly recommended* rather than *required*. I'm interpreting the first paragraph by adding the underlined clarifying words:

> Windows uses r11 for fast-walking of the stack frame. For more information, see the Stack Walking section. Because of this requirement in order for fast-walking to work, r11 must point to the topmost link in the chain at all times if you want fast-walking to work. If you know what's good for you, do not use r11 for general purposes—if you ignore this advice, then your code will not generate correct stack walks during analysis.

[3] It is totally not a coincidence that *lr* and *pc* are adjacent registers. This allows you to push a set of registers including *lr*, and then pop the same set of registers, but substituting *pc* for *lr*.

Raymond Chen

**Follow**