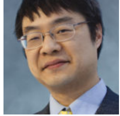
The Alpha AXP, part 15: Variadic functions

devblogs.microsoft.com/oldnewthing/20170828-00

August 28, 2017



Raymond Chen

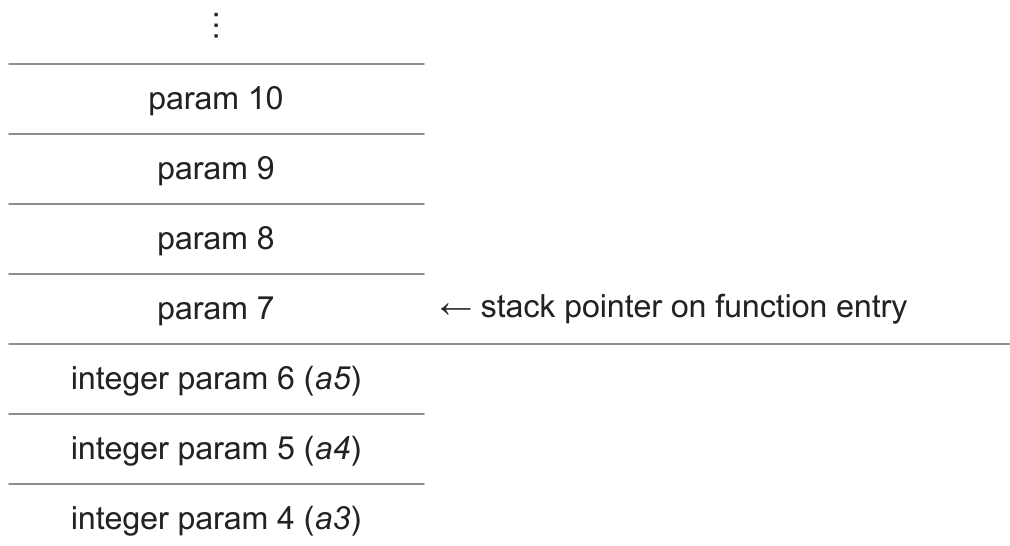
As noted in the initial plunge, the first six integer parameters are passed in registers, and the first six floating point parameters are passed in a different set of registers. So how does the callee known at function entry which registers to spill, and in what order?[1]

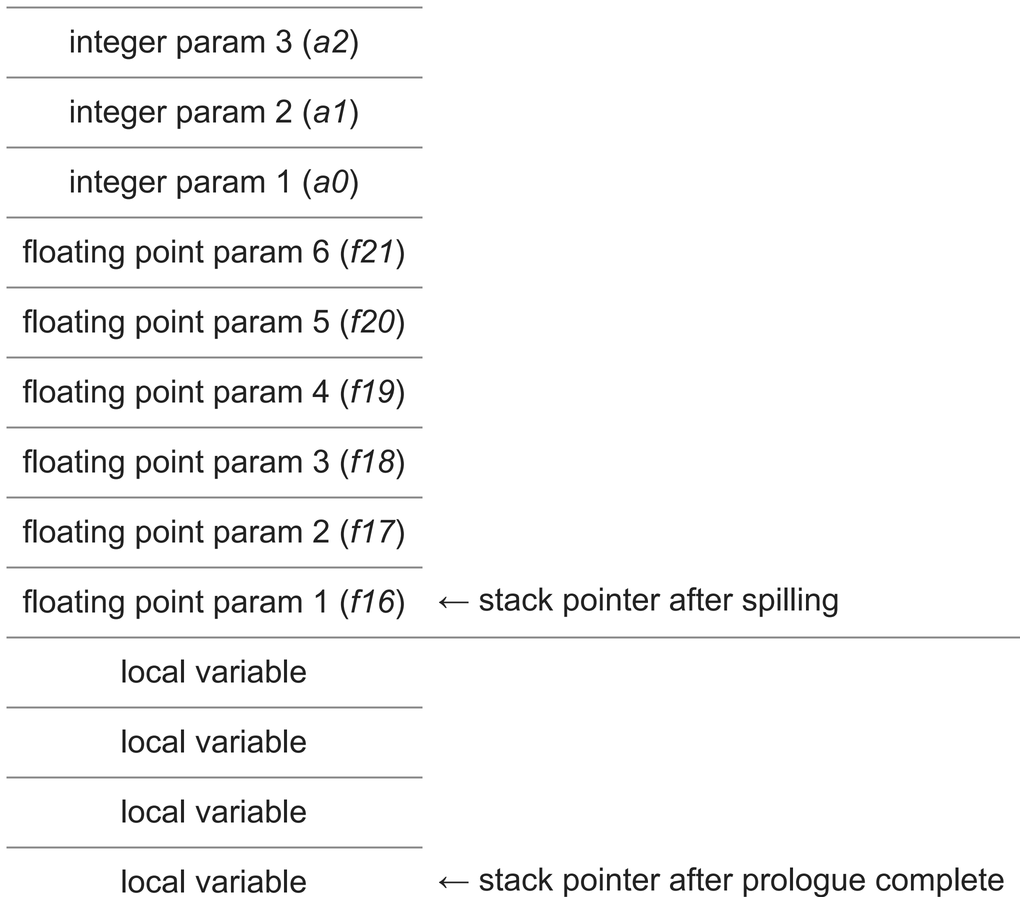
Answer: It doesn't. So it just spills everything.

First, a detail on the calling convention: The first six parameters are passed in registers, and if you pass a parameter in an integer register, then the corresponding floating point register is unused, and vice versa. In other words:

- The first parameter is passed in either *a0* or *f16*.
- The second parameter is passed in either *a1* or *f17*.
- …
- The sixth parameter is passed in either *a5* or *f21*.

On entry to a variadic function, the function spills all the integer parameter registers onto the stack first, and then spills the floating point parameter registers onto the stack next. The result is a stack that looks like this:





The `va_list` type is a structure:

```
typedef struct __va_list
{
  char* base;
  size_t offset;
} va_list;
```

The `va_start` macro initializes `base` to point to "integer param 1" and `offset` to 8 × the number of non-variadic parameters.

If you invoke the `va_arg` macro with a non-floating point type as the second parameter, then it operates in an unsurprising manner: It retrieves the data from `base + offset` and then increases the `offset` by the size of the data (rounded up to the nearest multiple of eight).

But invoking the `va_arg` macro with a floating point type as the second parameter is weirder: If the `offset` is less than 48, then it retrieves the data from `base + offset - 48`, resulting in a "reach-back" into the parallel array of spilled floating point registers. If the `offset` is greater than or equal to 48, then it retrieves the data from `base + offset` as usual. Regardless of where the data is read from, the `offset` increases by the size of the data (rounded up to the nearest multiple of eight).

The implementations of the `va_start` and `va_arg` macros take advantage of special-purpose compiler intrinsics that did a lot of the magic.

There are a few optimizations possible here. For one thing, the compiler doesn't need to spill non-variadic parameters, though it does need to reserve space for them on the stack so that the `va_arg` macro continues to work.[2] Furthermore, if the compiler can observe that `va_arg` is never invoked with a floating point type, then it doesn't need to spill the floating point registers at all. (Similarly, if `va_arg` is always invoked with floating point types, then the integer registers don't need to be spilled.)

I don't remember whether the Microsoft compiler actually implemented any of these optimizations.

[1] It turns out that this question is not Alpha-specific. It applies to any architecture that passes variadic parameters differently depending on their type.

[2] If the compiler can observe that `va_arg` is never invoked with a floating point type, then it doesn't even need to reserve space for the non-variadic parameters. It can just point the `base` at where the first integer parameter would have been, even though it now points into the local variables. Those local variables will never be read as parameters because the initial `offset` skips over them.

Raymond Chen

Follow