Inspecting Obj-C parameters in gdb

Since the addition of i386 and x86_64 to the Mac OS's repertoire several years back, remembering which registers are used for what has become difficult, and this can complicate the debugging of code for which you have no symbols. So here is my cheat-sheet (posted here, mostly so that I can find it again without google-ing for old mailing list posts; but, I figure someone else may find it useful as well):

arm (before prolog)

- `$r0` ➡ arg0 (self)
- `$r1` ➡ arg1 (_cmd)
- `$r2` ➡ arg2
- `$r3` ➡ arg3
- `*($sp)` ➡ arg4
- `*($sp+4)` ➡ arg5
- `*($sp+8)` ➡ arg6

ppc/ppc64

- `$r3` ➡ arg0 (self)
- `$r4` ➡ arg1 (_cmd)
- `$r5` ➡ arg2
- `$r6` ➡ arg3
- `$r7` ➡ arg4
- `$r8` ➡ arg5

i386 (before prolog)

- `*($esp+4n)` ➡ arg(n)
- `*($esp)` ➡ arg0 (self)
- `*($esp+4)` ➡ arg1 (_cmd)
- `*($esp+8)` ➡ arg2
- `*($esp+12)` ➡ arg3

- `*($esp+16)` ➡ arg4
- `*($esp+20)` ➡ arg5

i386 (after prolog)

- `*($ebp+8+4n)` ➡ arg(n)
- `*($ebp+4)` ➡ Return addr
- `*($ebp+8)` ➡ arg0 (self)
- `*($ebp+12)` ➡ arg1 (_cmd)
- `*($ebp+16)` ➡ arg2
- `*($ebp+20)` ➡ arg3
- `*($ebp+24)` ➡ arg4
- `*($ebp+28)` ➡ arg5
- `*($ebp)` ➡ Previous $ebp

x86_64

- `$rdi` ➡ arg0 (self)
- `$rsi` ➡ arg1 (_cmd)
- `$rdx` ➡ arg2
- `$rcx` ➡ arg3
- `$r8` ➡ arg4
- `$r9` ➡ arg5

So, if you have a method defined as:
`-(id)method:(id)foo bar:(id)bar baz:(id)baz`
you can print each of the parameters with:

	arm	ppc/ppc64	x86_64	i386 (before prolog)	i386 (after prolog)
self	`po $r0`	`po $r3`	`po $rdi`	`po *(id*) ($esp)`	`po *(id*) ($ebp+8)`
_cmd	`p (SEL)$r1`	`p (SEL)$r4`	`p (SEL)$rsi`	`p *(SEL*) ($esp+4)`	`p *(SEL*) ($ebp+12)`
foo	`po $r2`	`po $r5`	`po $rdx`	`po *(id*) ($esp+8)`	`po *(id*) ($ebp+16)`

bar	`po $r3`	`po $r6`	`po $rcx`	`po *(id*) ($esp+12)`	`po *(id*) ($ebp+20)`
baz	`po *(id*) ($sp)`	`po $r7`	`po $r8`	`po *(id*) ($esp+16)`	`po *(id*) ($ebp+24)`

As Blake mentioned in his comment, on i386, if you're at the beginning of a function or method, before the prolog has executed (i.e. the bit of code responsible for saving registers, adjusting the stack pointer, etc.), then ebp won't have been set up for you yet. So, I've amended the above table.

That complexity is another reason I long for the simplicity of PowerPC asm, not to mention M68k asm; at least x86_64 has made the step towards using registers for parameters where possible.

Edited to add: In case it isn't obvious, these particular stack offsets and registers assignments only make sense when dealing with pointer and integer parameters and return values. When structures and floating point values come into the mix, things can get more complicated.

Edited to add: I've added registers/stack offsets for arm. But note that these are for before the prolog has executed. Arm code seems much looser about what happens in its function prologs, so there really isn't a standard layout post-prolog

Filed under:
Uncategorized by Clark Cox