Compilers: Let Me Obfuscate That For You

27 Apr 2014

In my previous post, I had assembled the following C loop, using `gcc -O0`:

```
    for (i = 0; i < 10000000; i++) { }
```

It became this assembly:

```
[...]
    .L3:
        addl    $1, -4(%rbp)
    .L2:
        cmpl    $9999999, -4(%rbp)
        jle .L3
[...]
```

You might wonder why exactly the "`< 10000000`" test became a `jle` (jump less-than or equal) of 999999, instead of a `jl` (jump less-than) of 1000000, like the C code. I've always suspected it's a compiler optimization to save bytes based on different binary encodings.

Seems that it is (stackoverflow): it's about reducing the constant by 1 so that it can fit in fewer bytes. That's great, since such optimizations can improve the efficiency of the CPU caches, the cache hit ratio, and performance in general.

Out of curiosity I compiled both versions and ran objdump on the binaries, to see how many bytes were saved:

```
$ diff objdump.jl objdump.jle
124,125c124,125
<   4004c1: 81 7d fc 40 42 0f 00    cmpl   $0xf4240,-0x4(%rbp)
<   4004c8: 7c f3                   jl     4004bd
---
>   4004c1: 81 7d fc 3f 42 0f 00    cmpl   $0xf423f,-0x4(%rbp)
>   4004c8: 7e f3                   jle    4004bd
```

None. But the compiler did it anyway, obfuscating the binary output. I was happier when I didn't know this.

What's annoying is that this makes debugging harder that it needs to be. Constants can be a good target to search for when browsing long assembly listings. Had I searched for 1000000, I wouldn't have found it.

The compiler does know whether the optimization makes a difference or not – after all, *it's* the one emitting the final machine code!

This gcc version is:

```
$ gcc --version
gcc (GCC) 4.8.2 20131212 (Red Hat 4.8.2-7)
```

It does provide `-Og`. From the man page "-Og enables optimizations that do not interfere with debugging.", however, this still changes the constant.

I believe the code responsible in gcc is the simplify_compare_const() function from combine.c:

```
/* Do some canonicalizations based on the comparison code.  We prefer
     comparisons against zero and then prefer equality comparisons.
     If we can reduce the size of a constant, we will do that too.  */
  switch (code)
    {
    case LT:
      /* < C is equivalent to <= (C - 1) */
      if (const_op > 0)
        {
          const_op -= 1;
          code = LE;
          /* ... fall through to LE case below.  */
        }
      else
```

This can work; here's the optimization saving 3 bytes when the constant is 128:

```
jl of 128:
  4004c1:   81 7d fc 80 00 00 00    cmpl   $0x80,-0x4(%rbp)
  4004c8:   7c f3                   jl     4004bd

jle of 127:
  4004c1:   83 7d fc 7f             cmpl   $0x7f,-0x4(%rbp)
  4004c5:   7e f6                   jle    4004bd
```

Neat.

That makes me a bit happier about the obfuscation, knowing that some of the time this helps.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).