CS-118-02 Week 10

Varun Narravula

2022-04-04

You can follow along with the examples provided in the folders. I use Nix, and got a bit lazy when uploading, so you can look at instructions to build in the buildPhase portions of those files and ignore everything else.

Shift vs. Rotate

You've done bitwise shifts before in C and C++ using `<<` and `>>`. In assembly, their equivalents are `shl` (*shift left*) and `shr` (*shift right*). `rax >> 1` If you want to perform the equivalent of `$ax << 2` in assembly, you would write `shl ax, 2`; if you wanted to shift right by 2, then you do `shr ax, 2`.

However, what about rotating? Rotating is a bit like shifting, except that if any values go off the sides on a shift, they will be wrapped onto the other side. If you rotate a number `0b00001011` right by 1, then it would result in `0b10000101`, where the 1 that got shifted got put back on the other side instead of getting discarded.

Another instruction that looks similar to the normal shifts is `sal` (*shift arithmetic right*) and `sar` (*shift arithmetic left*). What does this do? Normal shift instructions only work on unsigned values, and overflow/underflow otherwise. Arithmetic shifts like these will make sure to preserve the sign.

Let's look at some code to see why this can be useful.

```
_start:
    mov ax, -1
    shl ax, 1
```

Test vs Cmp

There is another instruction called `test`, which works similarly to `cmp`. However, it performs a bitwise `and` (i.e. `test eax, eax` would compute `and eax, eax`) instead of subtracting (i.e. `cmp, eax, 0` would compute `sub eax, 0`) and set flags depending on the result. You can use this to compare as well.

Intrinsics

What are intrinsics? You already know how SSE can make things much more efficient because it is a SIMD instruction set. However, it's extremely difficult to do this using assembly; most people use intrinsics instead. It is the same instruction set, but it uses C, which is much easier than assembly because of the access to much more robust C tooling. Intel provides an SDK that interfaces with SSE instructions and looks very similar to actual assembly code; all you need to do is include a single header. There is no need to deal with compliated compile instructions anymore; all you need is a C compiler.