15-122: Principles of Imperative Computation

Recitation 3 — Josh Zimmerman

Hexadecimal notation

For an example, convert the hex number 0x7f2c to binary.

Solution: Based on the table, we know that 7 corresponds to 0111, f corresponds to 1111, 2 corresponds to 0010, and c corresponds to 1100.

So, 0x7f2c = 0111111100101100.

Bit manipulation

Let's look at some examples of masking so you can get a better idea of how it's used. First, let's write a function that, given a pixel in the ARGB format, returns the green and blue components of it. Your solution should use only `&`.

Solution:

```
1 typedef int pixel;
2 int greenAndBlue(pixel p)
3 //@ensures 0 <= \result && \result <= 0xffff;
4 {
5    // We only want the lower 16 bits of p, so bitwise-and the others with 0
6    // to set them all to 0
7    return p & 0xffff;
8 }
```

Now, let's write a function that gets the alpha and red pixels of a pixel in the ARGB format. Your solution can use any of the bitwise operators, but will not need all of them.

Solution:

```
1 typedef int pixel;
2 int alphaAndRed(pixel p)
3 //@ensures 0 <= \result && \result <= 0xffff;
4 {
5    // First, we want to put the top 16 bits in the bottom of the number.
6    // Then, we want to get rid of any sign extension that the right shift
7    // caused, so we use a mask to get rid of anything above the bottom 16 bits
8    return (x >> 16) & 0xffff;
9 }
```

Two's Complement

Now, let's formally prove that flipping the bits and adding 1 does, in fact, produce the negation. Work on this by yourself or with other people to prove this.

Solution: First, observe that for all `x`, `x + ~x == 11...11`. But when we add 1 to that, we simply get 0.

So, `x + ~x + 1 = 0`. With a bit of algebra, we can see that this means that `-x = ~x + 1`.

However, that only holds if `x` is not the minimum integer, since taking the negative of the minimum integer causes overflow. Thus, `-INT_MIN = INT_MIN`, and that is the one exception to this proof.