Data Project: Worked Example

Fall 2019

Here is an example of an integer puzzle. In this example, we will assume that you have solved the **`negate()`** and **`is_equal()`** puzzles.

```
/*
 * fits_bits - Return 1 if x can be represented as an
 *  n-bit two's complement integer, else return 0
 * 1 <= n <= 32
 * Examples: fits_bits(5, 3) = 0
 *           fits_bits(-4, 3) = 1
 * Legal ops:  ! ~ & ^ | + << >>
 * Max ops: 15
 * Rating: 4
 */

int fits_bits(int x, int n) {
      return 2;
}
```

For this puzzle, we want to tell if a number fits in n bits. What do we know about numbers which fit inside of n bits?

- For such a number x, $-2^{N-1} \le x \le (2^{N-1} - 1)$.
- Each bit from the nth upward is:q the same. (Remember that we can extend a signed n-bit
  integer to an m-bit integer by copying the sign bit into the upper m − n bits.)
- If we truncate the number to n bits, the result should represent the same number.

The first approach is difficult to do given the restrictions of this assignment. The second or third approach might be doable with bit level operations; let's try the third approach.

How can we truncate a 32-bit number to n bits? We need an operation which can cut off the upper bits of x. Fortunately, such an operation exists! A simple left shift by 32 − n will suffice. If x cannot fit within n bits, then truncating it will change the value of the represented number; if it can, then truncating it will not change the value.

Next we need a way to check that the result is the same as the original value of x. But since the truncated value is shifted, the bits do not align. We need to find a way to return the bits to their original position.

What if we right shift back by the same amount? Doing so will place the bits in the right place, and will sign-extend the truncated value back to 32 bits. This is exactly what we need.
Now that the bits are in the correct place, all that is needed is a way to check equality. Fortunately,
we can employ the same operations used from **`is_equal().`**
Putting this together, we get the following solution:

```
int fits_bits(int x, int n) {
      int shift = 32 + negate(n);
      int truncated = (x << shift) >> shift;
      return is_equal(x, truncated);
}
```

Note: **Do not actually call these functions in a real puzzle.** Instead, substitute the bit operations from the functions into the puzzle.