CS4400 *Alex Clemmer, u0458675*

Lecture 3 Problem Set

Problem 2.71

1. What is wrong with this code?

> This function guarantees in a lot of cases that the sign extension is not kept when we mask with 0xFF.

2. Give a correct implementation using only left and right shifts along with 1 subtraction.

```
int xbyte(packed_t word, int bytenum)
{
      /* Shift all the way to the right */
      int t = (3 - bytenum) << 3;

      /* Shift back all the way to the left */
      int32_t res = (word << ti) >> 24;
}
```

Problem 2.76

NOTE: I'll assume that the variable we're multiplying is called `x`.

1. K = 17

```
x = (x << 4) + x
```

2. K = -7

```
x = x - (x << 3)
```

3. K = 60

```
x = (x << 6) - (x << 2)
```

4. K = -112

```
x = (x << 4) - (x << 7)
```

Problem 2.81

1. `(x<y) == (-x>-y)`

   > Signed range is asymmetric, so supplying `x = INT_MIN` (or equivalent minimum value) and almost anything for `y` will probably break this system, as negating `INT_MIN` gives us `INT_MIN`, and nothing is strictly smaller than `INT_MIN`, which is required for the righthand expression to be true.

2. `((x+y)<<4) + y-x == 17*y+15*x`

   > True. `16*(x+y)+y-x == 16x-x+16y+y == 17y+15x`. There are no corner cases like there were in the last one.

3. `~x+~y+1 == ~(x+y)`

   > True. First, `~x+~y+1 == -x-1+(~y) == -x-1-y`. Then, `~(x+y) == -(x+y)-1 == -x-y-1`. So they are equivalent.

4. `(ux-uy) == -(unsigned)(y-x)`

   > False. If `x = -10` and `y = -1`, then `10-1 == 9` and `-(unsigned)(-10-(-1)) ==`

5. `((x >> 2) << 2) <= x`

   > True. The binary representation of a number increases monotonically as the number itself increases even in the case of negative numbers (*i.e.*, for any number n, the binary representation of $n+1$ is larger than n was) and therefore when you divide by two and multiply by a power of 2 (*e.g.*, $x = x/n * 2^n$), you will at best end up with the same number, and at worst, a smaller number.