RE #3

Task:

1. Figure out what the code does.
2. Some versions have the 0x1010101 constant, some do not. Why?

Procedure:

1)

Like challenge 2, we are performing a set of bitwise operations by shifting and applying a mask. For this I went straight to recreating the code functionality in python (see chall4\_script.py).

I first tested the test value 11254 (10101111110110) and got the output 00010000001010. I was a bit confused, so I tried some other test values. I tried a few random numbers before counting up from one before trying a few other values. From what we can see in the table it seems like the code is counting the number of ‘1’s in a binary string.

|  |  |  |
| --- | --- | --- |
| **Dec** | **Bin** | **Output** |
| 482 | 111100010 | 100000101 |
| 954 | 1110111010 | 1000000111 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0001 |
| 3 | 0011 | 0010 |
| 4 | 0100 | 0001 |
| 15 | 1111 | 0100 |
| 256 | 100000000 | 100000001 |
| 255 | 11111111 | 1000 |

However, the first few random values were giving odd results. It seems like numbers above 256 are giving flawed results. Why is this happening? It would seem like the line   
value = ((value & 0xf0f0f0f) \* 0x1010101)  
only allows us to count up to 8 bits. We would need to expand the code in order to count higher.

2)

Now we need to determine the purpose of the 0x1010101 constant. Looking at the other versions, they all perform a 24 bit shit operation. However, the versions without the constant does not perform the multiplication operation and instead do two additions. Performing the two operations and then comparing the values confirmed that these operations are equal.