Russian Peasant Multiplication

Given three positive integers **1 ≤ A, B, P ≤** $\mathbf{10^{18}}$ find **A*B % P**. That is we need to find remainder after division of **A** times **B** by **P**.

Simply multiplying these two numbers and taking the result modulo won't work, writing something like **printf("%lld\n", (A%P * B%P) % P)** won't suffice as intermediate results can be as large as $\mathbf{10^{36}}$, this obviously overflows any current data type. One solution is to write our own big integer arithmetic, but do we really want to do that for such a simple task? There must be a more efficient way!

Well, it turns out there is, the algorithm presented here solves this task in $\mathbf{O(\log A)}$ time. "Russian Peasant Multiplication Algorithm" exploits the simple fact of how we used to multiply numbers in elementary school.

Let's illustrate long multiplication on two numbers, written in binary base, so we might get some idea for the algorithm.

$$
\begin{array}{r}
\underline{101101 \cdot 10111} \\
1 \cdot \quad 10111 \\
0 \cdot \quad 101110 \\
1 \cdot \quad 1011100 \\
1 \cdot \quad 10111000 \\
0 \cdot \quad 101110000 \\
\underline{+1 \cdot 1011100000} \\
\quad 10000001011
\end{array}
$$

What's important to notice here is that each next row is obtained by shifting all bits in the current by one position to the left (which is equivalent to multiplying the previous number by two in base ten). For example 010111<<1 = 101110 (<<1 denotes shift left all bits by one), obviously 010111 is the same as 10111 since leading zeroes don't matter.

With this in mind we can devise the following algorithm:

1) *Iterate through all the bits of number **A** from right to left*
2) *If i-th bit is on (it's equal to 1) add the current result to the total*
3) *Do all of these operations modulo **P***

Since shifting to the left by one position is the same as multiplying by two, it's easy to see that we will avoid any overflows. Because $\mathbf{2*10^{18}}$ is well within the limits of 64bit integer type.