Handout: Why Binary Long Multiplication Works

Core idea: Binary long multiplication works because any binary number is a sum of powers of two, and multiplying by a power of two is just a left shift—so multiplication becomes "add the shifted copies of A for every 1-bit in B."

1) The identity behind long multiplication

Long multiplication (in any base) is an application of the distributive law:

$A \times (X + Y) = A \times X + A \times Y$.

- If you can express the multiplier as a sum of weighted digits, you can distribute A across that sum and add the partial products.

2) Write the multiplier as a sum of powers of two

Any binary number B can be written as:

$B = \Sigma (b_i \cdot 2^i)$, where each b_i is a bit (0 or 1).

This is just place value in base 2: each bit contributes either $0 \cdot 2^i$ or $1 \cdot 2^i$.

3) Distribute A across that sum

Using distributivity:

$A \times B = A \times \Sigma (b_i \cdot 2^i) = \Sigma (A \times b_i \cdot 2^i)$.

Because $b_i \in \{0,1\}$:

• If $b_i = 0$, the term contributes 0.

• If $b_i = 1$, the term contributes $A \times 2^i$.

4) Why shifting appears

In binary, multiplying by 2^i is precisely a left shift by i bits:

$A \times 2^i = A << i$.

So:

$A \times B = \Sigma\ (A << i)$ over all i where $b_i = 1$.

- This is the conceptual meaning of “shift-and-add.”

5) Why the ‘rows’ line up in the written method

Each row in long multiplication corresponds to one bit b_i:

• If $b_i = 0$, the row is all zeros.

• If $b_i = 1$, the row is A shifted left by i positions.

The shift aligns the partial product to the correct place value (2^i), just like decimal long multiplication aligns rows by powers of 10.

6) Why summing the rows gives the product

When you add all rows, you are summing all weighted contributions $A \cdot 2^i$ for the 1-bits of B.

- That sum equals A×B by the distributive law and base-2 place value.
- Therefore, binary long multiplication is correct.