The key trick of multiplication is memorizing a digit-to-digit table...
Everything else is just adding

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
</tr>
</tbody>
</table>
**Binary Multipliers**

The key trick of multiplication is memorizing a digit-to-digit table... Everything else is just adding

<table>
<thead>
<tr>
<th>×</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
</tr>
</tbody>
</table>

You’ve got to be kidding... It can’t be that easy
Binary Multiplication

Binary multiplication is implemented using the same basic longhand algorithm that you learned in grade school.

\[
\begin{array}{cccc}
A_3 & A_2 & A_1 & A_0 \\
x B_3 & B_2 & B_1 & B_0 \\
\end{array}
\]

\[
\begin{array}{cccc}
A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 \\
A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 \\
A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 \\
+ A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 \\
\end{array}
\]

Multiplying N-digit number by M-digit number gives (N+M)-digit result

Easy part: forming partial products (just an AND gate since \( B_1 \) is either 0 or 1)

Hard part: adding M, N-bit partial products
Multiplication: Implementation

1. Text Multiplicand
   - Shift left
   - 64 bits
   → 64-bit ALU

2. Shift right
   - Multiplier
   - 32 bits
   → Control test

3. Write
   - 64 bits

Start

1. Test Multiplier0
   - Multiplier0 = 1
   → 1a. Add multiplicand to product and place the result in Product register

2. Shift the Multiplicand register left 1 bit

3. Shift the Multiplier register right 1 bit

32nd repetition?
   - No: < 32 repetitions
   - Yes: 32 repetitions

Done
Second Version

- **Multiplicand**
  - 32 bits
  - 32-bit ALU
  - 32-bit ALU
  - Shift right
  - Write
  - Product
  - 64 bits

- **Multiplier**
  - Shift right
  - 32 bits

- **Control test**
  - Start
  - Multiplier0 = 0
  - Multiplier0 = 1

  1. **Test Multiplier0**
     - Multiplier0 = 1
     - Multiplier0 = 0

  1a. Add multiplicand to the left half of the product and place the result in the left half of the Product register

  2. Shift the Product register right 1 bit

  3. Shift the Multiplier register right 1 bit

  32nd repetition?
  - No: < 32 repetitions
  - Yes: 32 repetitions

  Done
## Example for second version

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Step</th>
<th>Multiplier</th>
<th>Multiplicand</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial</td>
<td>1011</td>
<td>0010</td>
<td>0000 0000</td>
</tr>
<tr>
<td>1</td>
<td>Test true shift right</td>
<td>1011 0101</td>
<td>0010</td>
<td>0010 0000 0001 0000</td>
</tr>
<tr>
<td>2</td>
<td>Test true shift right</td>
<td>0101 0010</td>
<td>0010</td>
<td>0011 0000 0001 1000</td>
</tr>
<tr>
<td>3</td>
<td>Test false shift right</td>
<td>0010 0001</td>
<td>0010</td>
<td>0001 1000 0000 1100</td>
</tr>
<tr>
<td>4</td>
<td>Test true shift right</td>
<td>0001 0000</td>
<td>0010</td>
<td>0010 1100 0001 0110</td>
</tr>
</tbody>
</table>
The trick is to use the lower half of the product to hold the multiplier during the operation.
What about the sign?

Positive numbers are easy.

How about negative numbers?
Faster Multiply

- A1 & B
- A0 & B
- A2 & B
- A3 & B
- A31 & B

- P0
- P1
- P2
- P31
- P32-P63
Division

1. Subtract Divisor from the Remainder leave the result in the Remainder

Test Remainder

>=0

Shift Quotient to the left set its rightmost bit = 1

Restored Remainder by adding Divisor Shift Quotient to the left set its rightmost bit = 0

Shift Divisor Register right 1 bit

Repeat 33 times