Karatsuba's Algorithm for multiplying 0111 * 1001 walk-though

Preliminaries

1. – Let's compute the answer in decimal so we know what to expect from the algorithm...

0111 (base 2) = 7 (base 10)
1001 (base 2) = 9 (base 10)

7 * 9 = 63

63 (base 10) = 111111 (base 2).

Therefore, the answer to 0111 * 1001 is 111111.

2. – The two numbers to be multiplied are denoted *a* and *b*. We always multiply two numbers of *n* bits each where *n* is a power of 2. If the two numbers are originally a different number of bits and/or are not a power of 2, we pad zeros on the left of the numbers as necessary. In particular, to compute the new number of bits *n′* of the two numbers, take the longer of the two original numbers, and compute the next largest power of 2. So if we have a 5 bit and a 3 bit number, then *n′* is 8. The extra bits of both numbers are zeros padded on the left to make them 8 bits each.

3. – The *n/2* high-bits (left-most bits) of *a* are denoted a1 and the *n/2* low-bits (right-most bits) of *a* are denoted *a0*. The same goes for *b*. So in this notation, the two numbers to be multiplied are shown below.

a1	*a0*
0 1	1 1

b1	*b0*
1 0	0 1

4. – Karatsuba's algorithm uses the following equation to compute *a*b* (which involves *n*-bit operands) in terms of 3 multiplications involving *n/2*-bit operands and a few additions and subtractions. Those 3 smaller multiplications are computed the same way by this equation below. This technique is called divide and conquer, i.e., reducing the problem into several smaller subproblems, recursively solving the subproblems and using those solutions to compute the solution for the original problem.

$c = c2 * 2^n + c1 * 2^{(n/2)} + c0$,

where $c2 = a1 * b1$, $c0 = a0 * b0$,
and $c1 = (a1 + a0) * (b1 + b0) - (c2 + c0)$.

Note that multiplying a binary number by *2^n* is equivalent to padding *n* bits on the right of the number (i.e., shifting the number left n times). And multiplying by *2^n(/2) i*s equivalent to padding *n/2* bits on the right.

Karatsuba(0111, 1001)

NOTE: The underlined multiplications are computed recursively. See their computation in the other recursive calls below. Calls involving two 1-bit numbers are not shown; those are handled by the base-case.

a1	*a0*
0 1	1 1

b1	*b0*
1 0	0 1

c2 = *a1* * *b1* = 01 * 10 = 10

c0 = *a0* * *b0* = 11 * 01 = 11

c1 = *(a1 + a0) * (b1 + b0) – (c2 + c0)* = 0100 * 0011 – 101 *// added 0s padding to the multiplicands*

= 1100 – 101 = 1100 – 0101 = 1100 + 1011 *// two's complement method*

= 0111

c = *c2 * 2^n + c1 * 2^(n/2) + c0* = (10) * 2^n + (0111) * 2^(n/2) + (11) =

= 100000 + 011100 + 11 = 111111

Answer: 111111

Karatsuba(01, 10)

a1	*a0*
0	1

b1	*b0*
1	0

c2 = *a1* * *b1* = 0 * 1 = 0

c0 = *a0* * *b0* = 1 * 0 = 0

c1 = *(a1* + *a0)* * *(b1* + *b0)* – *(c2* + *c0)* = 1*1 – (0 + 0)
= 1 – 0 = 1

c = *c2* * *2^n* + *c1* * *2^(n/2)* + *c0* = 000 + 10 + 0 = 010

Answer: 10 *// removed 0s from the left*

Karatsuba(11, 01)

a1	*a0*
1	1

b1	*b0*
0	1

c2 = *a1* * *b1* = 1 * 0 = 0

c0 = *a0* * *b0* = 1 * 1 = 1

c1 = (*a1* + *a0*) * (*b1* + *b0*) – (*c2* + *c0*)
= 10 * 01 – 1 *// added 0s padding to the multiplicands; see this call above, but reversed operands*
= 10 – 01 = 10 + 11 *// two's complement method*
= 01

c = *c2* * 2^*n* + *c1* * 2^(*n*/2) + *c0* = 000 + 010 + 1 = 011

Answer: 11

Karatsuba(0100, 0011)

a1	*a0*
0 1	0 0

b1	*b0*
0 0	1 1

*c2 = a1 * b1 =* 01 * 00 = 0 *// recursive call walk-though is omitted*

*c0 = a0 * b0 =* 00 * 11 = 0 *// recursive call walk-though is omitted*

*c1 = (a1 + a0) * (b1 + b0) – (c2 + c0) =* 01 * 11 – 0 *// see this call above, but reversed operands*
= 11 – 0 = 11 – 00 = 11 + 00 *// two's complement method*
= 11

*c = c2 * 2^n + c1 * 2^(n/2) + c0 =* 00000 + 1100 + 0 = 01100

Answer: 1100