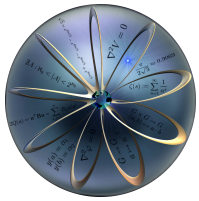

planetmath.org

Math for the people, by the people.

doubling and halving algorithm for integer multiplication

Canonical name	DoublingAndHalvingAlgorithmForIntegerMultiplication
Date of creation	2013-03-22 17:01:20
Last modified on	2013-03-22 17:01:20
Owner	CompositeFan (12809)
Last modified by	CompositeFan (12809)
Numerical id	5
Author	CompositeFan (12809)
Entry type	Algorithm
Classification	msc 00A06
Classification	msc 00A05
Classification	msc 11B25

Because multiplying and dividing by 2 is often easier for humans than multiplying and dividing by other numbers there is an algorithm for multiplication of any two integers that takes advantage of multiplication and division by 2.

Call the algorithm with two integers.

1. Use one of the integers to start a column on the left and the other to start a column on the right. (Either number can be put in either column, there are very minor optimizations that are unlikely to make a difference in performance, such as not choosing for the left column numbers that end long Cunningham chains).

2. Divide the previous integer on the left column by 2 and write the yield below it, ignoring any fractional part there may be. Multiply the previous integer on the right column by 2 and write the product below.

3. Repeat Step 2 until the yield on the left column is 1.

4. For every even number on the left column, cross out the right column's number of the same row.

5. Add up the the numbers on the right column that haven't been crossed out.

For example, to multiply 108 by 255:

108	~~255~~
54	~~510~~
27	1020
13	2040
6	~~4080~~
3	8160
1	16320
	27540

This works in any base (as long as one doesn't get confused about parity in odd bases). For example, 18 times 24 in base 5:

33	~~44~~
14	143
4	~~341~~
2	~~1232~~
1	3014
	3212

Naturally one might wonder if this can be applied to binary and used by computers. After all, halving and ignoring the fractional part is even easier: it's just a matter of shifting the bits to the right, and it doesn't matter what the computer does with the discarded bit (as long as it doesn't put it back into the original byte or word in the most significant bit or the sign bit). Doubling is also easy, just a shift left, with the only concern being overflow.

1010	~~1~~11
101	1110
10	~~1~~1100
1	111000
	1000110

Of course this algorithm is not suitable for large integer multiplication as is required in the search for large prime numbers.

References

[1] Paul Erdős & János Surányi *Topics in the theory of numbers* New York: Springer (2003): 5

[2] Ogilvy & Anderson, *Excursions in Number Theory*. Oxford: Oxford University Press (1966). Reprinted New York: Dover (1988): 6 - 8