# Modular Multiplication Without Trial Division 

By Peter L. Montgomery


#### Abstract

Let $N>1$. We present a method for multiplying two integers (called $N$-residues) modulo $N$ while avoiding division by $N$. $N$-residues are represented in a nonstandard way, so this method is useful only if several computations are done modulo one $N$. The addition and subtraction algorithms are unchanged.


1. Description. Some algorithms [1], [2], [4], [5] require extensive modular arithmetic. We propose a representation of residue classes so as to speed modular multiplication without affecting the modular addition and subtraction algorithms.

Other recent algorithms for modular arithmetic appear in [3], [6].
Fix $N>1$. Define an $N$-residue to be a residue class modulo $N$. Select a radix $R$ coprime to $N$ (possibly the machine word size or a power thereof) such that $R>N$ and such that computations modulo $R$ are inexpensive to process. Let $R^{-1}$ and $N^{\prime}$ be integers satisfying $0<R^{-1}<N$ and $0<N^{\prime}<R$ and $R R^{-1}-N N^{\prime}=1$.

For $0 \leqslant i<N$, let $i$ represent the residue class containing $i R^{-1} \bmod N$. This is a complete residue system. The rationale behind this selection is our ability to quickly compute $T R^{-1} \bmod N$ from $T$ if $0 \leqslant T<R N$, as shown in Algorithm REDC:

## function $\operatorname{REDC}(T)$

$m \leftarrow(T \bmod R) N^{\prime} \bmod R[$ so $0 \leqslant m<R]$
$t \leftarrow(T+m N) / R$
if $t \geqslant N$ then return $t-N$ else return $t$
To validate REDC, observe $m N \equiv T N^{\prime} N \equiv-T \bmod R$, so $t$ is an integer. Also, $t R \equiv T \bmod N$ so $t \equiv T R^{-1} \bmod N$. Thirdly, $0 \leqslant T+m N<R N+R N$, so $0 \leqslant t<$ $2 N$.

If $R$ and $N$ are large, then $T+m N$ may exceed the largest double-precision value. One can circumvent this by adjusting $m$ so $-R<m \leqslant 0$.

Given two numbers $x$ and $y$ between 0 and $N-1$ inclusive, let $z=\operatorname{REDC}(x y)$. Then $z \equiv(x y) R^{-1} \bmod N$, so $\left(x R^{-1}\right)\left(y R^{-1}\right) \equiv z R^{-1} \bmod N$. Also, $0 \leqslant z<N$, so $z$ is the product of $x$ and $y$ in this representation.

Other algorithms for operating on $N$-residues in this representation can be derived from the algorithms normally used. The addition algorithm is unchanged, since $x R^{-1}+y R^{-1} \equiv z R^{-1} \bmod N$ if and only if $x+y \equiv z \bmod N$. Also unchanged are
the algorithms for subtraction, negation, equality/inequality test, multiplication by an integer, and greatest common divisor with $N$.

To convert an integer $x$ to an $N$-residue, compute $x R \bmod N$. Equivalently, compute $\operatorname{REDC}\left((x \bmod N)\left(R^{2} \bmod N\right)\right)$. Constants and inputs should be converted once, at the start of an algorithm. To convert an $N$-residue to an integer, pad it with leading zeros and apply Algorithm REDC (thereby multiplying it by $R^{-1} \bmod N$ ).

To invert an $N$-residue, observe $\left(x R^{-1}\right)^{-1} \equiv z R^{-1} \bmod N$ if and only if $z \equiv$ $R^{2} x^{-1} \bmod N$. For modular division, observe $\left(x R^{-1}\right)\left(y R^{-1}\right)^{-1} \equiv z R^{-1} \bmod N$ if and only if $z \equiv x(\operatorname{REDC}(y))^{-1} \bmod N$.

The Jacobi symbol algorithm needs an extra negation if $(R / N)=-1$, since $\left(x R^{-1} / N\right)=(x / N)(R / N)$.

Let $M \mid N$. A change of modulus from $N($ using $R=R(N))$ to $M$ (using $R=R(M))$ proceeds normally if $R(M)=R(N)$. If $R(M) \neq R(N)$, multiply each $N$-residue by $(R(N) / R(M))^{-1} \bmod M$ during the conversion.
2. Multiprecision Case. If $N$ and $R$ are multiprecision, then the computations of $m$ and $m N$ within REDC involve multiprecision arithmetic. Let $b$ be the base used for multiprecision arithmetic, and assume $R=b^{n}$, where $n>0$. Let $T=$ $\left(T_{2 n-1} T_{2 n-2} \cdots T_{0}\right)_{b}$ satisfy $0 \leqslant T<R N$. We can compute $T R^{-1} \bmod N$ with $n$ single-precision multiplications modulo $R, n$ multiplications of single-precision integers by $N$, and some additions:

```
\(c \leftarrow 0\)
for \(i:=0\) step 1 to \(n-1\) do
        \(\left(d T_{i+n-1} \cdots T_{i}\right)_{b} \leftarrow\left(0 T_{i+n-1} \cdots T_{i}\right)_{b}+N^{*}\left(T_{i} N^{\prime} \bmod R\right)\)
        \(\left(c T_{i+n}\right)_{b} \leftarrow c+d+T_{i+n}\)
        [ \(T\) is a multiple of \(b^{i+1}\) ]
    \(\left[T+c b^{i+n+1}\right.\) is congruent \(\bmod N\) to the original \(\left.T\right]\)
    \(\left[0 \leqslant T<\left(R+b^{i}\right) N\right]\)
end for
if \(\left(c T_{2 n-1} \cdots T_{n}\right)_{b} \geqslant N\) then
    return \(\left(c T_{2 n-1} \cdots T_{n}\right)_{b}-N\)
else
    return \(\left(T_{2 n-1} \cdots T_{n}\right)_{b}\)
end if
```

Here variable $c$ represents a delayed carry-it will always be 0 or 1 .
3. Hardware Implementation. This algorithm is suitable for hardware or software. A hardware implementation can use a variation of these ideas to overlap the multiplication and reduction phases. Suppose $R=2^{n}$ and $N$ is odd. Let $x=$ $\left(x_{n-1} x_{n-2} \cdots x_{0}\right)_{2}$, where each $x_{i}$ is 0 or 1 . Let $0 \leqslant y<N$. To compute $x y R^{-1} \bmod N$, set $S_{0}=0$ and $S_{i+1}$ to $\left(S_{i}+x_{i} y\right) / 2$ or $\left(S_{i}+x_{i} y+N\right) / 2$, whichever is an integer, for $i=0,1,2, \ldots, n-1$. By induction, $2^{i} S_{i} \equiv\left(x_{i-1} \cdots x_{0}\right) y \bmod N$ and $0 \leqslant S_{i}<N+y<2 N$. Therefore $x y R^{-1} \bmod N$ is either $S_{n}$ or $S_{n}-N$.

1. J. M. Pollard, "Theorems on factorization and primality testing," Proc. Cambridge Philos. Soc., v. 76, 1974, pp. 521-528.
2. J. M. Pollard, "A Monte Carlo method for factorization," BIT, v. 15, 1975, p. 331-334.
3. George B. Purdy, "A carry-free algorithm for finding the greatest common divisor of two integers," Comput. Math. Appl. v. 9, 1983, pp. 311-316.
4. R. L. Rivest, A. Shamir \& L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Comm. ACM, v. 21, 1978, pp. 120-126; reprinted in Comm. ACM, v. 26, 1983, pp. 96-99.
5. J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," J. Assoc. Comput. Mach., v. 27, 1980, pp. 701-717.
6. Gustavus J. Simmons, "A redundant number system that speeds up modular arithmetic," Abstract 801-10-427, Abstracts Amer. Math. Soc., v. 4, 1983, p. 27.
