

REIST Division: An Implementation-Oriented Framing of Centered Remainder Arithmetic for Modular Addition

Rudolf Stepan
Independent Researcher
ORCID: 0009-0004-2842-2579

Abstract—Centered remainder representations express signed cyclic corrections using established residue classes. This note defines the integer interval and its bounded modular-addition update, including odd moduli, midpoint conventions and overflow prerequisites. It retains historical ARMv8-A and x86-64 measurements against compiler-remainder baselines. These measurements have not been repeated after correcting boundary handling and do not establish a benefit over bounded non-negative modular addition, which also avoids division. A new controlled evaluation is required before making a representation-specific performance claim. The method concerns addition-dominated updates and does not replace general division or multiplication-oriented Montgomery and Barrett reduction.

Index Terms—REIST, Integer Division, Modular Arithmetic, Centered Remainder, Cryptographic Workloads, ARMv8-A, x86-64.

CORRECTION AND SCOPE (24 OCTOBER 2026)

The benchmark tables below are retained as historical observations, not as a fresh evaluation of the corrected formulas. Earlier versions mishandled some odd-modulus boundaries. Reproducible performance claims require corrected kernels, exact revision identifiers, and a bounded non-negative baseline. For integer $B > 0$, let $l = -\lfloor B/2 \rfloor$ and $h = \lfloor (B-1)/2 \rfloor$. If $a, b \in [l, h]$, compute $s = a + b$ and apply $s \leftarrow s - B$ if $s > h$, or $s \leftarrow s + B$ if $s < l$. One correction suffices. Arbitrary inputs need normalization first. The sum, correction, and shifted construction must be representable without signed overflow. For non-negative $a, b < B$, the alternative $s = a + b$ followed by subtraction of B when $s \geq B$ also avoids division. Either bounded rule can use a runtime modulus. Source-level conditionals do not guarantee branchless machine code or constant-time execution.

I. INTRODUCTION

A. Domain and intended use of REIST

REIST Division admits direct implementation in low-level integer code. The second half of this work therefore treats REIST not merely as a mathematical object, but as a concrete arithmetic primitive. Measurements are reported from an open-source benchmark suite that implements REIST side-by-side with classical modulo operations and evaluates them on both ARMv8-A and x86-64 processors to test whether the theoretical advantages translate into real hardware performance.

While many examples in this report are drawn from lattice-based cryptography and modular arithmetic, REIST Division is not a cryptography-specific construction. Its core domain is integer arithmetic with signed remainders and the modeling of cyclic and feedback-driven processes where “too much” and “too little” are symmetric correction states. Cryptographic workloads are used primarily as a stress test and as a convenient source of high-volume modular operations.

In high-performance cryptographic libraries, modular multiplication is typically accelerated by Montgomery or Barrett reduction. The benchmarks in this work do not compete with such schemes directly; instead, they explore how a signed-remainder correction rule behaves in generic modular workloads. For multiplication-heavy workloads, REIST should be viewed as complementary rather than as a replacement for Montgomery or Barrett reduction.

The goal of the comparison against the compiler-provided modulo operator is to isolate the architectural effect of replacing division-based remainder handling by a signed correction rule, not to claim superiority over state-of-the-art modular reduction techniques.

II. THEORETICAL FRAMEWORK

A. Classical Division

Given integers T and $B > 0$, classical division provides unique integers q and r satisfying

$$T = qB + r, \quad 0 \leq r < B. \quad (1)$$

This representative convention is useful for indexing; centered representatives can be useful for signed cyclic differences. Neither convention changes the underlying quotient ring.

B. Definition of REIST Division

We define the REIST remainder of an integer T with respect to a positive modulus B as the unique integer r such that

$$T = qB + r, \quad q \in \mathbb{Z}, \quad -\frac{B}{2} \leq r < \frac{B}{2}. \quad (2)$$

REIST therefore uses a centered, half-open remainder interval $[-B/2, B/2)$. An exact-arithmetic construction, with non-negative mathematical modulo, is

$$r = ((T + \lfloor B/2 \rfloor) \bmod B) - \lfloor B/2 \rfloor, \quad (3)$$

which guarantees that r always lies in the desired interval and that $+B/2$ never occurs.

C. Relation to Rounding

For every integer $B > 0$, the REIST quotient satisfies

$$q = \left\lfloor \frac{T}{B} + \frac{1}{2} \right\rfloor. \quad (4)$$

This is nearest-integer rounding with ties toward positive infinity; ties can occur only for even B . REIST differs in that the remainder is treated as a persistent signed correction variable that can be fed back into iterative update loops.

III. BENCHMARK SUMMARY

TABLE I
ARMV8-A SPEEDUP RANGES ACROSS TESTED MODULI.

Workload	O3	SIMD
Modular addition	2.1–2.3×	4.1–4.4×
Polynomial modular addition	4.3–4.5×	4.3–4.5×
Remainder only	$\approx 1.0\times$	$\approx 1.0\times$
Hash mixing	0.79–0.80×	0.79–0.83×

TABLE II
X86-64 SPEEDUP RANGES ACROSS TESTED MODULI.

Workload	O3	SIMD
Modular addition	7.9–8.9×	7.1–8.7×
Polynomial modular addition	12.4–12.8×	13.1–17.0×
Remainder only	$\approx 1.0\times$	$\approx 0.9\times$
Hash mixing	0.80–0.81×	0.73–0.75×

Interpretation: The historical comparison concerns particular implementations and compiler-remainder baselines. It does not isolate a benefit of centered representatives. ARX arithmetic already wraps modulo a power of two without explicit division; the cause of hash-mixing slowdowns has not been isolated.

IV. CONCLUSION

REIST Division uses centered representatives to express signed cyclic corrections. This is an established remainder convention, with application-dependent advantages and conversion costs.

The historical results motivate controlled remeasurement. They do not establish a speedup over bounded non-negative modular addition. The corrected mathematical contribution is a precise centered-remainder invariant and its bounded update rule.

REFERENCES

- [1] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.
- [2] I. Niven, H. S. Zuckerman, and H. L. Montgomery, *An Introduction to the Theory of Numbers*, 5th ed. New York, NY, USA: Wiley, 1991.
- [3] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2012.
- [4] Repository (benchmark sources): <https://github.com/rudolfstefan/reist-crypto-bench>