RISC-V Word-Size Modular Instructions for Residue Number Systems

Laurent-Stéphane Didier, Jean-Marc Robert

RISC-V word-size modular instructions for Residue Number Systems

Laurent-Stéphane Didier and Jean-Marc Robert

IMATH, Université de Toulon, Toulon, France
laurent-stephane.didier@univ-tln.fr, jean-marc.robert@univ-tln.fr

Abstract. Residue Number Systems (RNS) are parallel number systems that allow the computation on large numbers. They are used in high performance digital signal processing devices and cryptographic applications. However, the rigidity of instruction set architectures of the market-dominant microprocessors limits the use of such number systems in software applications.
This article presents the impact of word-size modular arithmetic specific RISC-V instructions on the software implementation of Residue Number Systems. We evaluate this impact on several RNS modular multiplication sequential algorithms. We observe that the fastest implementation uses the Kawamura *et. al.* base extension. Simulations of architectures with GEM5 simulator show that RNS modular multiplication with Kawamura's base extension is 2.76 times faster using specific word-size modular arithmetic instructions than pseudo-Mersenne moduli for In Order processors. It is more than 3 times for Out of Order processors. Compared to x86 architectures, RISC-V simulations show that using specific instructions requires 4.5 times less cycles in In Order processors and 8 less in Out of Order ones.

Keywords: High performance number system, Residue Number Systems, modular multiplication, word-size modular arithmetic, RISC-V ISA

1 Introduction

This paper deals with modular multiplication of large numbers, and its applications. This operation is widely used in cryptographic computations such as RSA encryption/decryption, Elliptic Curve Cryptography, and more recently, the SIKE post-quantum protocol. Implementations of this operation aim to provide fast and secure computation. In the context of software implementation, the state-of-the-art cryptographic libraries such as OpenSSL make use of multi-precision approaches. In this work, we explore the use of the non-positional number system Residue Number Systems (RNS) in this context. We target the RISC-V platform and software implementations of this approach in order to evaluate the efficiency and the potential improvement offered by extension of the instruction set, in particular specific word-size modular arithmetic instructions. This research is based upon simulation of RISC-V platforms, and is prelude to future hardware implementations and experiments on real platforms.

Residue Number Systems (RNS) are non-weighted carry-free number systems which arithmetic is done over parallel finite rings. Such systems can be used in high performance signal processing [12], fault-tolerant computing [39], convolutional neural networks [42] and cryptographic applications such as RSA cryptosystems [9], homomorphic cryptography [8], elliptic curves [3]. These systems also have interesting leak-resistant properties [7] [26]. Because of their parallel property, they are suitable for vector implementations. However, the software implementation of RNS applications suffers from the cost of the word-size modular arithmetic [14].

RISC-V is an open standard Instruction Set Architecture (ISA) that aim to provide a free and open ISA suitable for processor designs. It was pioneered at University of California at Berkeley in 2010. Now, this initiative is supported by the RISC-V association that regroups many industrials and academics interested in such a collaboration. The RISC-V standards organization continuously introduces new ISA extensions to meet the needs of advanced computing. The ISA itself is designed according to the reduced instruction set computing (RISC) principles. It consists of a small base integer instruction set with several sets of modular extensions (integer multiplication, floating point operations, etc.). This ISA also has a dedicated space for future or custom extensions. There is a lot of novel research for customized extensions of RISC-V for specific scenarios [13].

This offers the opportunity to introduce new arithmetic-related instructions. In the context of post-quantum cryptography, some extensions for lattice-based crypto-protocols have been proposed to improve the NTT computation [17] [22], LAC scheme [16] or for computations in finite fields [2]. The extension of ISA can also be useful for more *exotic* number systems. Some extensions have been proposed for POSIT arithmetic [41] [27] and bit-slicing computations [24].

In this article we show the impact of word-size modular arithmetic instructions on the RNS operations. We target the modular multiplication which is a frequent and expensive operation in several cryptographic [28]. In RNS, this operation requires the conversion between two bases that can be computed through several ways.

Related works As mentioned above, RNS are widely used in various contexts in hardware implementations and are based on word-size modular operations [29]. This is motivated by the inherent property of these systems to be parallelizable. In our case, since we are targeting software implementations, there are very few works in this topic. Some improvements to the software implementation of word-size modular operations have been proposed to use actual processor arithmetic units [31] [4]. Some of these improvements are also useful in lattice-based cryptosystems using Number Theoretic Transform (NTT) [20] [21]. However the word-size modular operations remain an bottleneck in software implementations.

A comparative study that highlights this phenomenon has been proposed by Didier *et al.* in [14]. Their work explores the software RNS implementation of modular multiplication for various modulus precision, from 400 to 3251 bits, on `x86-64` platform. This work compares sequential (using the classical instruction set) and parallel (`AVX512`) implementations with the multi-precision `GMP`

library [19]. This work concludes on the costly impact of the word-size modular operation and its penalty on the performance. Our work presented here attempts to address this issue on the software sequential RNS implementations, on RISC-V platforms.

Contributions We evaluate the benefit of the use of processors having specific instructions for the word-size modular operation and compare it with regular software implementation. We target RISC-V ISA. Our evaluations depend on several parameters:

- the word-size modular operation method for the elementary RNS operations,
- the conversion methods between two RNS bases,
- the size of the RNS base,
- the RISC-V processor configuration,
- the cost of RISC-V specific word-size modular operation instructions.

We ran nearly 3,000 simulations with the GEM5 simulator which is a modular platform for computer-system architecture research [10].

Paper organisation The background on RNS and the RNS modular multiplication are reminded in section 2. Our new instructions are described in section 3, the experiment parameters are described in section 4. We show the experimental results in section 5.

2 Residue Number System

Residue Number Systems are non positional integer number systems that are based on the Chinese Remainder Theorem [18, 25, 40]. In such a system, an integer x is represented by its remainders $x_i = x \bmod m_i$. The values m_i are relatively prime numbers. The set $\mathcal{B}_m = \{m_1, m_2, \ldots, m_n\}$ forms the RNS base composed of n channels. The moduli m_i are usually chosen with the width w that corresponds to the target architecture word-size. We denote $\boldsymbol{M}$ their product. The advantage of such a number system is that additions, subtractions and multiplications can be performed in parallel on each channel:

$$z_i = x_i \odot y_i \bmod m_i \text{ where } \odot \in \{+, -, \times\}$$

Conversions The forward conversion to RNS is simply a modular operation on each base channel. The backward conversion can be done through different ways. The Chinese Remainder Theorem provides a computation formula in the target number system [25]:

$$x = \left| \sum_{i=1}^{n} x_i \left(\frac{\boldsymbol{M}}{m_i} \right)_{m_i}^{-1} \boldsymbol{M}_i \right|_{\boldsymbol{M}} = \sum_{i=1}^{n} x_i \left(\frac{\boldsymbol{M}}{m_i} \right)_{m_i}^{-1} \boldsymbol{M}_i - k \cdot \boldsymbol{M} \tag{1}$$

where

$$\boldsymbol{M}_i \times \left(\frac{\boldsymbol{M}}{m_i} \right)_{m_i}^{-1} \equiv 1 \pmod{\boldsymbol{M}}$$

The main drawback of this approach is that the values used in this sum are large.

An other method consists of the conversion into the Mixed Radix System. This requires modular computations on w-bit integers only. In this positional system, an integer x_{MRS} is as follows:

$$x_{MRS} = x'_0 + x'_1 m_0 + x'_2 m_0 m_1 + \cdots + x'_{n-1} \prod_{i=0}^{n-2} m_i$$

This conversion requires $\mathcal{O}(n^2)$ operations on w-bit operands and needs $\mathcal{O}(n^2)$ constants [38].

A trade-off between these two methods has been proposed by Kawamura *et al.* [23]. It is based on equation (1) and consists of the estimation of k with approximate values through $\mathcal{O}(n)$ operations on small values with $\mathcal{O}(n)$ constants. The approximate values are w-bit integers.

Base extension The base extension is the conversion of an RNS number from one RNS base to another. This consists of a backward conversion and a forward conversion to the targeted RNS base. Both operations are interleaved in order to minimize storage of intermediate values.

The first base extension has been proposed by Szabo and Tanaka. It is based on the mixed-radix conversion [38]. Shenoy and Kumaresan suggested to compute the value k in equation (1) using an extra modulus m_e [37]. If k is known, then it is possible to compute equation (1) in the target RNS base. Similarly, Kawamura *et al.* [23] proposed a conversion method based on their approximation of k.

RNS Modular multiplication In RNS, the modular multiplication is derived from the Montgomery multiplication [30] and requires base extensions [6, 32]. It is summarized in Algorithm 1. In [9], the authors remark that if the dynamic range of base $\mathcal{B}_m$ is large enough, then it is not necessary to completely compute the first base extension which can be approximated. For the second one, they use the Shenoy-Kumaresan method [37]. In the multiplication described in [23] both extensions are Kawamura's.

In our implementations of Algorithm 1, we chose $\mathcal{B}_m$ and $\mathcal{B}_{m'}$ in order to use the Bajard-Imbert [9] first extension at step 3. For the second extension at step 7, we use the Szabo-Tanaka method [38] or Kawamura *et al.* method [23].

3 New instructions for word-size modular arithmetic

The RISC-V project aim to provide an open RISC instruction set architecture for processor design. This project was started in 2010 at the University of California at Berkeley. Compared to x86 and ARM ISA, the RISC-V Foundation allows some customizations in the ISA specification. Some opcodes are reserved for custom instructions.

Algorithm 1 RNS Modular Multiplication

Require: x in $\mathcal{B}_m$ and $\mathcal{B}_{m'}$; y in $\mathcal{B}_m$ and $\mathcal{B}_{m'}$ such that $x < 2p$ and $y < 2p$.
Precomputation: $-p^{-1}$ in $\mathcal{B}_m$; p in $\mathcal{B}_{m'}$; $\boldsymbol{M}^{-1}$ in $\mathcal{B}_{m'}$
Ensure: $z = x \times y \times \boldsymbol{M}^{-1} \bmod p$ in $\mathcal{B}_m$ and $\mathcal{B}_{m'}$ such that $z < 2p$.
1: $s \leftarrow x \times y$ in $\mathcal{B}_{m'}$ and $\mathcal{B}_m$
2: $t \leftarrow s \times (-p^{-1})$ in $\mathcal{B}_m$
3: Base extension of t from $\mathcal{B}_m$ to $\mathcal{B}_{m'}$
4: $u \leftarrow t \times p$ in $\mathcal{B}_{m'}$
5: $v \leftarrow s + u$ in $\mathcal{B}_{m'}$
6: $w \leftarrow v \times \boldsymbol{M}^{-1}$ in $\mathcal{B}_{m'}$
7: Base extension of w from $\mathcal{B}_{m'}$ to $\mathcal{B}_m$
8: **return** w

In the RISC-V instruction set, the arithmetic operations are performed register to register. In the instruction format, the fields related to the register are always at the same place. The input registers are denoted `rs` and the output register is denoted `rd`. In this format, the seven least significant bits encode the instruction *opcode*. The standard provides for *custom* opcodes [44] that we are using in our proposition.

We propose three new instructions for modular addition, subtraction and multiplication. The modular operations we implement require three inputs: the two operands and the modulus. Similarly to multiply-add vector instructions [5], we use a third input register `rs3` which field is at bit 27-31 in the format instruction. The formats used for our word-size modular arithmetic instructions are summarized in Fig. 1.

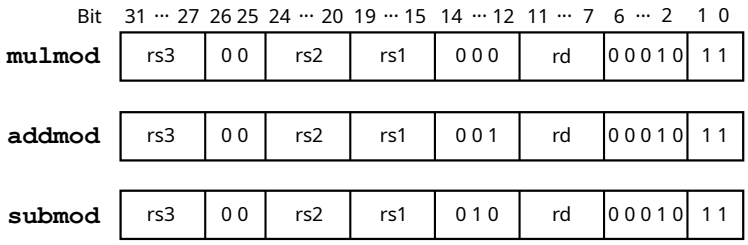

Fig. 1: The instructions format for word-size modular arithmetic

We provide hereafter the corresponding instruction operations:

1. `mulmod rd, rs1, rs2, rs3`
   $rd \leftarrow rs1 \times rs2 \bmod rs3$
2. `addmod rd, rs1, rs2, rs3`
   $rd \leftarrow rs1 + rs2 \bmod rs3$

3. `submod rd, rs1, rs2, rs3`
   $rd \leftarrow rs1 - rs2 \bmod rs3$

These instructions are used through intrinsic C functions. As an example, Figure 2 shows the intrinsic function used for the modular addition. Our new instructions have been added to the `gcc` cross-compiler from the RISC-V GNU Compiler Toolchain [35]. We also checked our compiled code with the Spike RISC-V ISA Simulator [35].

```
{C}
inline static int64_t addmod(int64_t a,
                  int64_t b, int64_t m)
{
  int64_t res;
  asm volatile
  (
    "addmod   %[z], %[w], %[x], %[y]\n\t"
    : [z] "=r" (res)
    : [w] "r" (a), [x] "r" (b), [y] "r" (m)
  );
  return res;
}
```

Fig. 2: Intrinsic C function for word-size modular addition

4 Experiments

We used and adapted the RNS library from [14] that we compiled with `gcc` with the `-O3` option. The binaries have been run within the GEM5 simulator and the number of cycles have been counted with the `rdcycle` for the RISC-V architecture and `rdtsc` for the x86 architectures. We used the evaluation protocol described in [14].

4.1 Simulator

The GEM5 simulator is an open source computer architecture cycle-level full-system simulator [10]. It is composed of a simulator core and parameterized models for a wide number of components such as in-order and out-of-order processors, DRAM or cache memories. It has been designed in order to effectively and efficiently emulate the behavior of modern processors. Amongst the various instruction set architectures, RISC-V [36] and x86 ISA are available [10].

This simulator appears to be accurate enough in order to quickly evaluate ISA extension, architectural choices and discard or select solutions for further investigations [1, 11, 15, 33, 34, 43]. We used GEM5 version 22.1 for our experiments.

4.2 GEM5 simulator parameters

The simulated RISC-V ISA uses 64-bit words. We chose a two levels 4-way associative cache architecture. The L1 cache is split in Data and Instruction parts of 32kb each. The L2 cache is set to 256kb. The simulated processors can access 2Gb dual channel DDR4_2400 RAM. The clock frequency is set to 1Ghz. These parameters are the same for all our simulations.

The GEM5 simulator offers In Order (Minor) and Out of Order (O3) processors. Both models allow to tune some parameters such as the operators, their delay or, in case of Out of Order processors, their number. The Minor model is a four-stage pipeline In Order processor, while the O3 model is a five-stage Out of Order processor. The operators are pipelined, except for the divisor and modulo.

4.3 Evaluated functions

We used several algorithmic parameters in our evaluation. The number of RNS channels is between 8 and 64 in steps of 8, which corresponds to moduli from 512 up to 4096 bits for the RNS modular multiplication. The word-size modular operations have been implemented with three different modulo operator (mod.):

1. The naive implementation provided by the C language, that is:

   ```
   {C}
           int64_t res, a, m;
           res = a%m;
   ```

   In case of processors without `DIV` instruction, i.e. our situation, the code will be compiled as a sequence of instructions computing a full division, and this is very costly.
   In the sequel, this version is named *Modulo* or *mod*.

2. Pseudo-Mersenne moduli (PM) [31]:
   In this case, the moduli are of the form $m_i = 2^w - c_i$ where w is the word size in bits and $c_i << 2^w$. Thus, one has $2^w \equiv c_i \bmod m_i$, and writing $a = a_l + 2^w a_h$ leads to $a \equiv a_l + c_i a_h \bmod m_i$. Though $c_i a_h$ may overflow 2^w bit, it is enough to repeat this process three times to get a fully reduced mod m_i value of a (see [31]).
   This corresponds to the Algorithm 2 for the modular reduction.

   This approach takes advantage of the special form of the moduli. It is much more efficient than the previous case, trading the computation of a full division by three multiplications by the small constant c_i and a few additions, shifts and masking operations.
   In the sequel, this version is named *Pseudo-Mersenne* or *PM*.

Algorithm 2 Pseudo-Mersenne modular reduction

Require: $a = a_l + a_h \times 2^w$ and c_i, precomputed $mask = 2^w - 1$
Ensure: $r \leftarrow a \bmod m_i$
1: $up \leftarrow a_h$
2: $lo \leftarrow a_l$
3: $t \leftarrow c_i \times up$
4: $up_2 \leftarrow t >> w$ // right w-bit shift
5: $lo_2 \leftarrow t\&mask$ // w-bit masking
6: $t \leftarrow t + lo + lo_2 + c_i \times up_2$
7: $up_3 \leftarrow t >> w$
8: $lo_3 \leftarrow t\&mask$
9: $t \leftarrow lo_3 + c_i \times up_3$
10: **return** $r \leftarrow t$

3. Using our new instructions (Inst.):
   In this case, the cost of the modular computation corresponds to the one of the single corresponding instruction.
   In the sequel, this version is named *Instruction* or *Inst.*

The evaluated RNS modular multiplication is described in Algorithm 1. In this algorithm, the most expensive function is the base extension. We have tested two variants of the RNS modular multiplication. While we use the Bajard-Imbert [9] first extension at step 3 for both variants, the second base extension at step 7 is either the Szabo-Tanaka in the first one [38] or the Kawamura *et al.* method in the second one [23].

In the sequel, the first variant is named *Szabo-Tanaka* or ST and the second variant is named *Kawamura* or K. These parameter abbreviations are used in the next figures of section 5.

This leads to a total of six configurations:

- Modulo Szabo-Tanaka
- Modulo Kawamura
- Pseudo-Mersenne Szabo-Tanaka
- Pseudo-Mersenne Kawamura
- Instruction Szabo-Tanaka
- Instruction Kawamura

4.4 Experimentation parameters

To evaluate the impact of the use of specific modular operation instructions on RNS, we carried out simulations varying some parameters. We evaluated the combination of the algorithmic parameters described in the previous paragraph. The number of RNS channels ranges from 8 to 64 by 8 steps. The delay of the integer ALU is set to 1. As a consequence the additions are computed with a delay of 1. The delays of the integer multiplier unit varies from 3 to 4. They are

set between 2 and 4 for the modular adder and between 4 and 9 for the modular multiplier, which is referred to as *long delays* case in the sequel. Finally, we simulated In Order (IO) and Out of Order (OoO) processor models.

5 Results

In this section, we first provide a global overview of the simulations. We afterward present the results of the simulations fetching the configurations mentioned in section 4.

Processor	Modular op. Base extension	Modular op. Base extension	Ratio
	`mulmod` delay: 4	`addmod` delay: 2	
IO	Modulo Szabo-Tanaka	Instruction Kawamura *et al.*	7.49
IO	Modulo Szabo-Tanaka	Instruction Szabo-Tanaka	5.46
IO	Modulo Szabo-Tanaka	Pseudo-Mers. Szabo-Tanaka	2.08
IO	Pseudo-Mers. Szabo-Tanaka	Instruction Szabo-Tanaka	2.63
IO	Pseudo-Mers. Szabo-Tanaka	Pseudo-Mers. Kawamura *et al.*	1.30
IO	Pseudo-Mers. Kawamura *et al.*	Instruction Kawamura *et al.*	2.76
IO	Instruction Szabo-Tanaka	Instruction Kawamura *et al.*	1.37
OoO	Modulo Szabo-Tanaka	Instruction Kawamura *et al.*	25.79
OoO	Modulo Szabo-Tanaka	Instruction Szabo-Tanaka	19
OoO	Modulo Szabo-Tanaka	Pseudo-Mers. Szabo-Tanaka	4.53
OoO	Pseudo-Mers. Szabo-Tanaka	Instruction Szabo-Tanaka	4.19
OoO	Pseudo-Mers. Szabo-Tanaka	Pseudo-Mers. Kawamura *et al.*	1.86
OoO	Pseudo-Mers. Kawamura *et al.*	Instruction Kawamura *et al.*	3.06
OoO	Instruction Szabo-Tanaka	Instruction Kawamura *et al.*	1.37
	`mulmod` delay: 9	`addmod` delay: 4	
IO	Pseudo-Mers. Szabo-Tanaka	Instruction Szabo-Tanaka	1.87
IO	Pseudo-Mers. Kawamura *et al.*	Instruction Kawamura *et al.*	1.92
OoO	Pseudo-Mers. Szabo-Tanaka	Instruction Szabo-Tanaka	2.74
OoO	Pseudo-Mers. Kawamura *et al.*	Instruction Kawamura *et al.*	2.30

Table 1: Speed ratio of 64-channel RNS modular multiplication with several word modular operations and base extensions, In Order (IO), Out of Order (OoO) RISC-V

5.1 Overview of the simulations

Table 1 shows the main and most significant results. The table is organised as follows:

– The first rows consider the fastest delays used in our experiment for the word-size operations. The `mulmod` delay is 4, the `addmod` delay is 2.

  - In the In Order processor model, we first give the range of speed-ups for various configurations. For example, the ratio between the slowest version (Modulo, Szabo-Tanaka) and the fastest (Inst. Kawamura *et al.*) is 7.49.
  - In the Out Of Order processor model, the performance hierarchy remains the same, however, the ratios are greater. The maximum speed-up is now 25.79, for the same versions as previously.
- The last rows consider the longest delays for the word-size operations: `mulmod` delay: 9, `addmod` delay: 4. The ratios are lower, however, even in this case, the benefit of the specific word-size modulo instructions remains significant:
  - The speed-up ratios of the Pseudo-Mersenne word-size modular reduction over the Instruction versions are nearly 2, in In Order processor model.
  - The speed-up ratio between the Pseudo-Mersenne word-size modular reduction and the Instruction versions, in Out of Order processor model reaches 2.74 with the Szabo and Tanaka version.

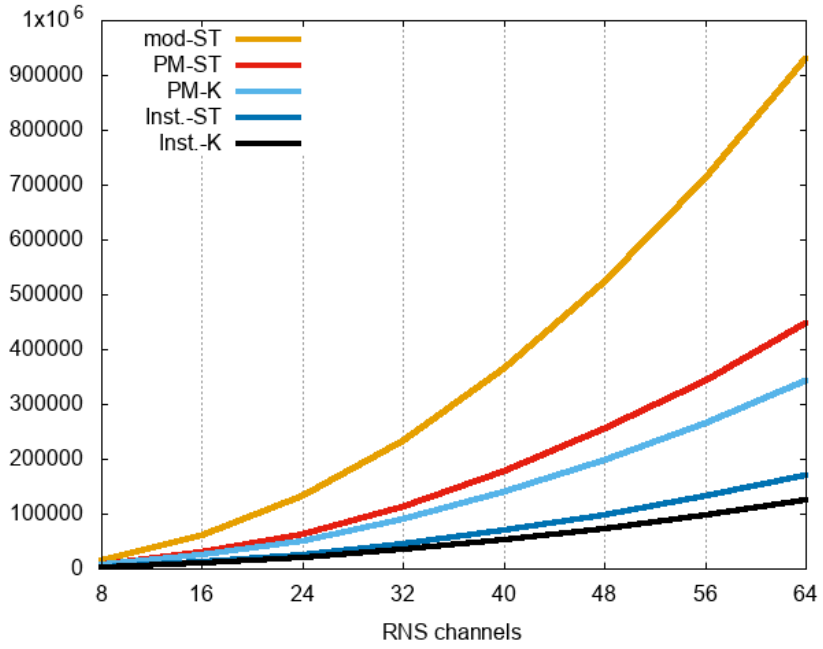

Fig. 3: RNS modular multiplication timing in clock cycle number, with In Order RISC-V model, `mulmod` delay: 4, `addmod` delay: 2

We do not provide all the timing values in clock cycle number. Nevertheless, Figure 3 gives an overview of the orders of magnitude for the slowest architecture configuration, which is the RISC-V In Order processor model. The general

quadratic behavior of Algorithm 1 is observed for all configurations. In case of 64 RNS channels (4096-bit RNS modular multiplication) and for one single RNS modular multiplication, the slowest version (Modulo, Szabo-Tanaka) takes 928804 clock cycles and the fastest (Inst. Kawamura *et al.*) takes 149096 clock cycles (`mulmod` delay: 4, `addmod` delay: 2).

5.2 RNS Modular multiplication algorithms

We first compare the RNS modular multiplication algorithms which mainly depend on the base extension methods. We tested two variants for the second base extension at step 7 of Algorithm 1: the Szabo-Tanaka (ST) [38] and the Kawamura *et al.* (K) [23] methods.

The Figure 3 summarizes the timing results expressed in clock cycles number for the In Order processor model. Without surprise, the measured delays regularly depend on the number of RNS channels for all tested algorithmic parameter combinations.

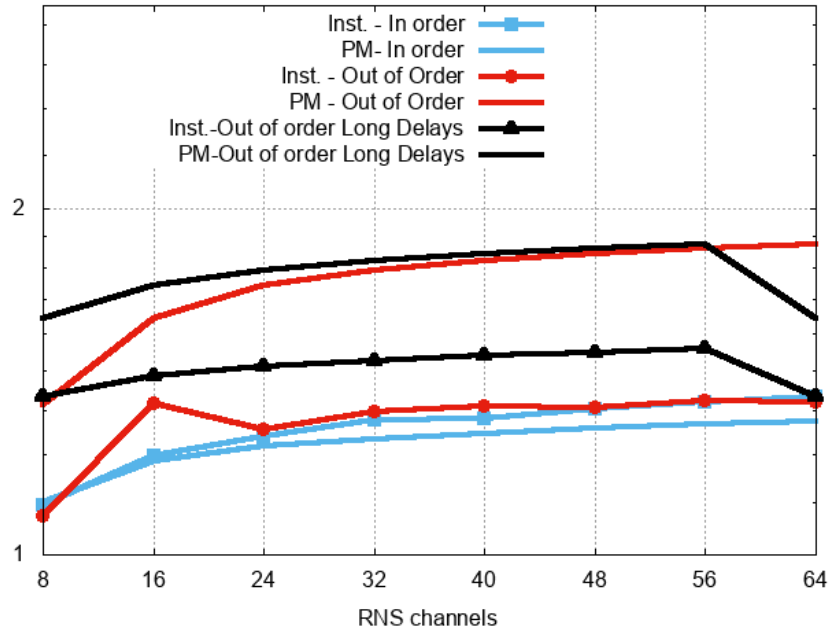

Fig. 4: RNS modular multiplication Speed-Up, comparison of Kawamura *et al.*'s versus Szabo and Tanaka methods

Figure 4 highlights the comparison between the base extension methods for In Order and Out of Order processors with `mulmod` delay=4 and `addmod` delay=2 in

blue and red lines respectively. The results of Out of Order processors with longer delays (`mulmod` delay=9 and `addmod` delay=4) are drawn in black. This figure shows the speed-up of the Kawamura's over Szabo-Tanaka method. It reaches a maximum of 1.86 in the pseudo-Mersenne case, Out of order processor. The speed-ups seem to have a weak correlation with the number of RNS channels.

The first part of the table 2 shows the speed-ups for 64-channel RNS focusing on the base extension comparison. The fastest base extension is the Kawamura's method regardless of the method used for the word-size modular operations. Using the same modular operation method, the use of Kawamura's base extension is more than 30% faster than Szabo-Tanaka's.

Finally, the best implementation that does not use our instructions is the combination of pseudo-Mersenne and Kawamura *et al.*'s method. We did not implemented the improvement of the word-size modular arithmetic provided in [31] [4], but the benefit seems to be close to the use of pseudo-Mersenne moduli.

5.3 Word-size operations

We now compare the performances of the RNS modular multiplication with respect to the word-size operation, in case of In Order processor model.

Figure 5 summarizes the results in terms of relative speed-ups. Considering the versions with the C language modulo operation and comparing it with our instruction equipped processor implementation, the speed-ups are weakly correlated with the RNS channel number.

The use of the C modulo operation gives the slowest configurations. This operation is basically a division which is the slowest arithmetic operator implemented in the processors. The modular reduction with pseudo-Mersenne is always slower than the versions using our instructions for word-size modular operations. We notice that:

- with the delays for word-size operations `addmod`=2, `mulmod`=4, the best speed-up over pseudo-Mersenne is of 2.76. It is achieved with the Kaxamura *et al.*'s configuration, while the gain in the Szabo-Tanaka case is slightly below, see red lines.
- with the *long delays* case (`addmod`=4, `mulmod`=9), the speed-ups are around 1.8 at best, see black lines.
- the improvement provided by our instructions over pseudo-Mersenne is similar in the Szabo-Tanaka and the Kawamura *et al.* case (see red curves). We observe the same in the *long delays* simulations (black curves).

The second part of the table 2 shows the speed-ups for 64-channel RNS focusing on the benefit of the versions using our instructions for word-size modular operations. Whatever the architecture version (In or Out of Order, *long delays*), the speed-ups range from 1.87 (Szabo and Tanaka, Pseudo-Mersenne versus Inst., In Order and slowest delays) up to 19 (Szabo and Tanaka, Modulo versus Inst.). The most significant speed-ups are between the fastest conventional

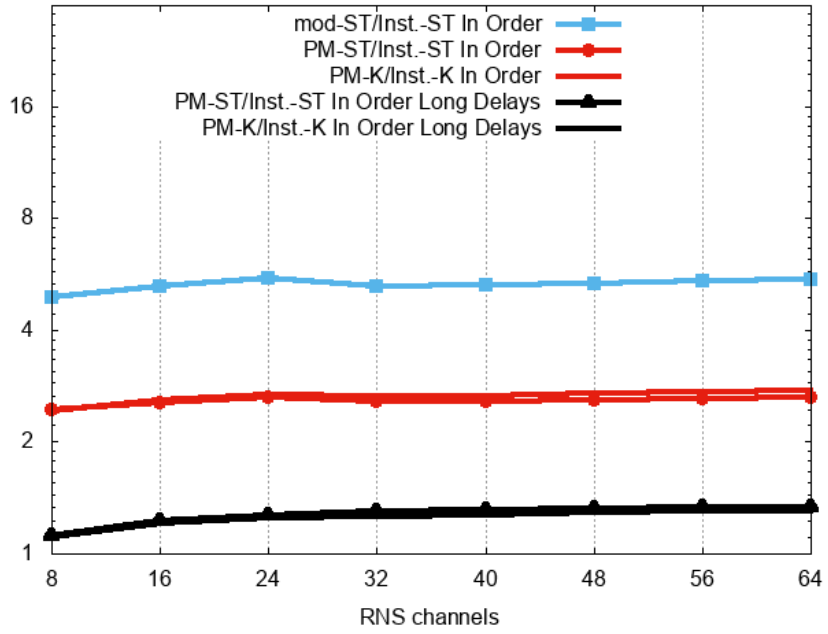

Fig. 5: RNS modular multiplication Speed-Up of Inst. over the C modulo operation and pseudo-Mersenne reduction (PM), In Order processor model

version (Pseudo-Mersenne and Kawamura) and the fastest with the word-size modular operation Instruction (Kawamura again):

- RISC-V Out of Order `mulmod` delay: 4 `addmod` delay: 2, the speed-up is 3.06;
- RISC-V Out of Order `mulmod` delay: 9 `addmod` delay: 4, the speed-up is 2.30.

These values show the interest of the implementation of the word-size modular operation Instructions in our simulation, and this is the motivation to continue this work in future hardware implementations.

5.4 Architectures comparison

Figure 6 shows the relative speed-ups of various combinations of modular operation and base extension methods on Out of Order processors. It highlights the improvement provided by the use of the modular instructions `addmod` and `mulmod`.

Out of order processors Out of Order processors offer the opportunity to compute simultaneously in several processing units. This leads to a better use of the hardware and as a consequence, a better use of our instructions. For instance, modular multiplication with word-size modular operation performed with our

Base extension comparison		
RISC-V In Order `mulmod` delay: 4 `addmod` delay: 2		
Mod. Operation	Compared base extension methods	Ratios
Pseudo-Mersenne	Szabo-Tanaka vs. Kawamura *et al.*	1.30
Instruction		1.37
Modulo operation comparison		
Base extension	Compared modulo operation methods	Ratios
RISC-V In Order `mulmod` delay: 4 `addmod` delay: 2		
Szabo-Tanaka	Modulo vs. Instruction	5.46
Szabo-Tanaka	Pseudo-Mersenne vs. Instruction	2.63
Kawamura *et al.*	Pseudo-Mersenne vs. Instruction	2.76
RISC-V In Order `mulmod` delay: 9 `addmod` delay: 4		
Szabo-Tanaka	Pseudo-Mersenne vs. Instruction	1.87
Kawamura *et al.*	Pseudo-Mersenne vs. Instruction	1.92
RISC-V Out of Order `mulmod` delay: 4 `addmod` delay: 2		
Szabo-Tanaka	Modulo vs. Instruction	19
Szabo-Tanaka	Modulo vs. Pseudo-Mersenne	4.53
Szabo-Tanaka	Pseudo-Mersenne vs. Instruction	4.19
Kawamura *et al.*	Pseudo-Mersenne vs. Instruction	3.06
RISC-V Out of Order `mulmod` delay: 9 `addmod` delay: 4		
Szabo-Tanaka	Pseudo-Mersenne vs. Instruction	2.74
Kawamura *et al.*	Pseudo-Mersenne vs. Instruction	2.30

Table 2: Speed ratio of 64-channel RNS modular multiplication with several word modular operations and base extensions, In Order, Out of Order RISC-V processors

instructions and Kawamura's base extension is 25.79 times faster than using the modulo operator and Szabo-Tanaka base extension (see Tab. 1). Compared to the fastest implementation that uses pseudo-Mersennes moduli and Kawamura's base extension, the implementation with our modular operation instructions is 3.06 times faster (Tab. 2).

Long delays case Our simulations are based on the assumption that word-size modular additions and multiplications can be computed with a delay of 2 and 4 cycles respectively. Although we believe that these delays are feasible, they could be much longer in a real implementations. Thus, we have estimated the delay of RNS modular multiplication using word-size modular additions and multiplications with a delay of up to 4 and 9, respectively.

In both cases, the speed gain through the use of our instructions remains interesting (see Tab. 2). Compared to the pseudo-Mersenne and Kawamura's case, the timing improvement for In Order processors is 92%, and more than 2 times faster in the Out of Order case.

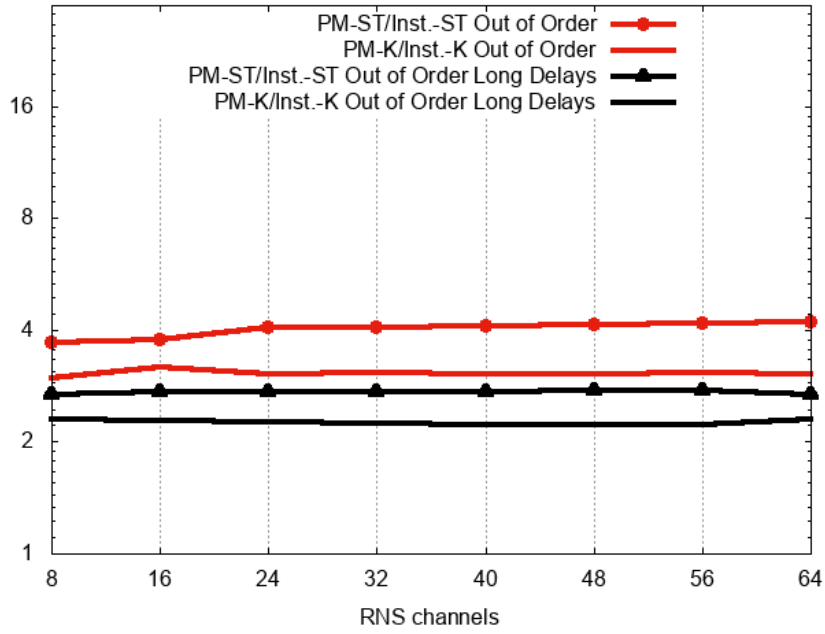

Fig. 6: RNS modular multiplication Speed-Up of Inst. over the Pseudo-Mersenne reduction (PM), Out of Order processor model

Comparison with x86 As a sake of comparison, we also simulated implementations of the RNS modular multiplication compiled for the Intel x86 ISA. The simulated architectures have the same cache, memory and frequency parameters. Both RISC-V and x86 architecture have two integer ALU of delay 1 and one integer multiplier of delay 3. The RISC-V has one modular adder of delay 2 and one modular multiplier of delay 4. The x86 ISA does not make use of specific word-size modular operator. The x86 binaries have been compiled with the same version of `gcc` and the same options than for the RISC-V target. We evaluated the timings for both In Order and Out of Order GEM5 processor models.

Table 3 summarizes the cycles required for RNS modular multiplication in both architectures using various word-size modular operation and base extension methods. These results cover both In Order and Out of Order processors.

In this experiment, it can be observed that the fastest RISC-V implementation of the RNS modular multiplication with our modular operation instructions requires fewer cycles than the fastest x86 implementation. In the case of In Order processors, it takes 4.5 times less cycles. In the Out of Order case, the ratio is 8 times less.

Without using our word-size modular instructions, the best combination is to use pseudo-Mersenne moduli and Kawamura *et al.* base extension. In this

Proc.	Mod. operations Base extension	Cycles
In Order		
x86	Pseudo-Mersenne Szabo-Tanaka	892302
x86	Pseudo-Mersenne Kawamura *et al.*	615359
RISC-V	Pseudo-Mersenne Szabo-Tanaka	446603
RISC-V	Instruction Szabo-Tanaka	177255
RISC-V	Pseudo-Mersenne Kawamura *et al.*	342415
RISC-V	Instruction Kawamura *et al.*	135682
Out of Order		
x86	Pseudo-Mersenne Szabo-Tanaka	484142
x86	Pseudo-Mersenne Kawamura *et al.*	361170
RISC-V	Pseudo-Mersenne Szabo-Tanaka	249718
RISC-V	Instruction Szabo-Tanaka	59548
RISC-V	Pseudo-Mersenne Kawamura *et al.*	134120
RISC-V	Instruction Kawamura *et al.*	43865

Table 3: Number of cycles of 64-channel RNS modular multiplication with several word modular operations and base extensions, RISC-V and x86 processors

case, the RISC-V RNS modular multiplication needs 79% less cycles in In Order processors. It is 3.61 times faster for Out of Order processors.

6 Conclusion

In this paper, we have presented the impact on the performance of RISC-V dedicated word-size modular operation instructions for RNS modular multiplication with large modulus. We have studied the following configurations:

- 2 modular multiplication variants:
  - Szabo-Tanaka
  - Kawamura
- 3 word size modular operation variants:
  - "C" compiled modulo
  - Pseudo-Mersenne moduli
  - our proposed new RISC-V Instructions, with two variants of delay
- 2 RISC-V configurations
  - In Order processor
  - Out Of Order processor

These combinations have been used to experiment with sequential RNS modular multiplications whose precision ranges from 8 to 64 64-bit RNS channels, or 512 to 4096 bits for the modulo size. The simulations were performed with the GEM5 simulator. The total number of simulations is almost 3000. We measured the performance in clock cycles for each of these configurations. We also ran the corresponding implementation on the x86-64 GEM5 simulator.

The use of specific instructions for word-size modular operations greatly improves the Residue Number Systems computation speed on both In Order and Out of Order RISC-V processors. The benefit is greater on Out of Order architectures due to the parallel nature of Residue Number Systems. Compared to the fastest implementation of RNS modular multiplication using pseudo-Mersenne moduli and the Kawamura *et al.* base extension, using our instructions yields implementations that are up to 3 times faster. The gain remains important for word-size modular operators with long delays, even with modular multiplier that has twice the delay of multipliers.

This shows that the use of specific instructions for word-size modular operations has a significant impact on the performance of the software implementation of RNS operations.

Future work This result motivates future hardware implementations of word-size modular operators for RISC-V processors.

We will also extend the simulation exploration to other configurations such as investigating different instruction strategies that use three instead of four registers. In the medium term, we also plan to implement corresponding vector instruction sets to take advantage of the natural ability to parallelization of the RNS. Our goal is to achieve, if possible, competitive performance levels for software RNS implementations on RISC-V-like platforms. This may also enable secure and randomized implementations of large modular operations for cryptographic use cases.

References

1. Ayaz Akram and Lina Sawalha. Validation of the gem5 simulator for x86 architectures. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 53–58. IEEE, 2019.
2. Erdem Alkim, Hülya Evkan, Norman Lahr, Ruben Niederhagen, and Richard Petri. ISA extensions for finite field arithmetic accelerating kyber and newhope on RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3), 2020.
3. S. Antão, J.-C. Bajard, and L. Sousa. RNS based elliptic curve point multiplication for massive parallel architectures. *The Computer Journal*, 55(5):629–647, 2012.
4. Daichi Aoki, Kazuhiko Minematsu, Toshihiko Okamura, and Tsuyoshi Takagi. Efficient word size modular multiplication over signed integers. In *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*, pages 94–101. IEEE, 2022.
5. Krste Asanovi. *RISC-V "V" Vector Extension, Version 0.9-draft-1535cc0*. EECS Department, University of California, Berkeley, 2019.
6. J-C Bajard, L-S Didier, and Peter Kornerup. An RNS montgomery modular multiplication algorithm. *IEEE Transactions on Computers*, 47(7):766–776, 1998.
7. J.-C. Bajard, S. Duquesne, and M. Ercegovac. Combining leak-resistant arithmetic for elliptic curves defined over f_p. *Publications Mathématiques de Besançon. Algrèbre et Théorie des Nombres*, pages 67–87, 2013. ISSN: 1958-7236.
8. J.-C. Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. A full RNS variant of fv like somewhat homomorphic encryption schemes. In *SAC 2016, Selected Areas in Cryptography, St. John's, Newfoundland and Labrador, Canada*, 2016.

9. J.-C. Bajard and L. Imbert. A full RNS implementation of RSA. *IEEE Transactions on Computers*, 53(6):769–774, 2004.
10. Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7, 2011.
11. Anastasiia Butko, Rafael Garibotti, Luciano Ost, and Gilles Sassatelli. Accuracy evaluation of gem5 simulator system. In *7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC)*, pages 1–7. IEEE, 2012.
12. Chip-Hong Chang, Amir Sabbagh Molahosseini, Azadeh Alsadat Emrani Zarandi, and Tian Fatt Tay. Residue number systems: A new paradigm to datapath optimization for low-power and high-performance digital signal processing applications. *IEEE Circuits and Systems Magazine*, 15(4):26–44, 2015.
13. Enfang Cui, Tianzheng Li, and Qian Wei. RISC-V instruction set architecture extensions: A survey. *IEEE Access*, 11:24696–24711, 2023.
14. Laurent-Stéphane Didier, Jean-Marc Robert, Fangan Yssouf Dosso, and Nadia El Mrabet. A software comparison of RNS and PMNS. In *2022 IEEE 29th Symposium on Computer Arithmetic (ARITH)*, pages 86–93. IEEE, 2022.
15. Fernando A Endo, Damien Couroussé, and Henri-Pierre Charles. Micro-architectural simulation of in-order and out-of-order arm microprocessors with gem5. In *2014 international conference on embedded computer systems: Architectures, modeling, and simulation (SAMOS XIV)*, pages 266–273. IEEE, 2014.
16. Tim Fritzmann, Georg Sigl, and Johanna Sepúlveda. Extending the RISC-V instruction set for hardware acceleration of the post-quantum scheme LAC. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1420–1425. IEEE, 2020.
17. Tim Fritzmann, Georg Sigl, and Johanna Sepúlveda. RISQ-V: Tightly coupled RISC-V accelerators for post-quantum cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 239–280, 2020.
18. H. L. Garner. The residue number system. *IRE Transactions on Electronic Computers*, EL 8(6):140–147, 1959.
19. Torbjörn Granlund and al. GNU multiple precision arithmetic library 6.1.2. `https://gmplib.org/`.
20. Junhao Huang, Jipeng Zhang, Haosong Zhao, Zhe Liu, Ray CC Cheung, Çetin Kaya Koç, and Donglong Chen. Improved plantard arithmetic for lattice-based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):614–636, 2022.
21. Junhao Huang, Haosong Zhao, Jipeng Zhang, Wangchen Dai, Lu Zhou, Ray CC Cheung, Cetin Kaya Koc, and Donglong Chen. Yet another improvement of plantard arithmetic for faster kyber on low-end 32-bit iot devices. *arXiv preprint arXiv:2309.00440*, 2023.
22. Emre Karabulut and Aydin Aysu. RANTT: A RISC-V architecture extension for the number theoretic transform. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 26–32. IEEE, 2020.
23. Shinichi Kawamura, Masanobu Koike, Fumihiko Sano, and Atsushi Shimbo. Cox-rower architecture for fast parallel montgomery multiplication. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 523–538, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

24. Pantea Kiaei, Thomas Conroy, and Patrick Schaumont. Architecture support for bitslicing. *IEEE Transactions on Emerging Topics in Computing*, 11(2):497–510, 2023.
25. Donald E Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.
26. Andrea Lesavourey, Christophe Negre, and Thomas Plantard. Efficient leak resistant modular exponentiation in rns. In *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)*, pages 156–163. IEEE, 2017.
27. David Mallasén, Raul Murillo, Alberto A Del Barrio, Guillermo Botella, Luis Piñuel, and Manuel Prieto-Matias. PERCIVAL: open-source posit RISC-V core with quire capability. *IEEE Transactions on Emerging Topics in Computing*, 10(3):1241–1252, 2022.
28. Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.
29. PV Ananda Mohan, PK Meher, and T Stouraitis. *Arithmetic circuits for DSP applications*, chapter RNS-Based arithmetic circuits and applications, pages 186–236. John Wiley & Sons, 2017.
30. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.
31. Thomas Plantard. Efficient word size modular arithmetic. *IEEE Transactions on Emerging Topics in Computing*, 9(3):1506–1518, 2021.
32. Karl C Posch and Reinhard Posch. Modulo reduction in residue number systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(5):449–454, 1995.
33. Yudi Qiu, Tao Huang, Yuxin Tang, Yanwei Liu, Yang Kong, Xulin Yu, Xiaoyang Zeng, and Yibo Fan. Gem5tune: A parameter auto-tuning framework for gem5 simulator to reduce errors. *IEEE Transactions on Computers*, 2023.
34. Hemendra Rawat and Patrick Schaumont. Vector instruction set extensions for efficient computation of keccak. *IEEE Transactions on Computers*, 66(10):1778–1789, 2017.
35. riscv collab. RISC-V GNU compiler toolchain. `https://github.com/riscv-collab/riscv-gnu-toolchain`, 2022.
36. Alec Roelke and Mircea R Stan. Risc5: Implementing the RISC-V ISA in gem5. In *First Workshop on Computer Architecture Research with RISC-V (CARRV)*, volume 7, 2017.
37. A.P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computers*, 38(2):292–297, 1989.
38. Nicholas S Szabo and Richard I Tanaka. *Residue arithmetic and its applications to computer technology*. New York: McGraw-Hill, 1967.
39. Thian Fatt Tay and Chip-Hong Chang. *Embedded systems design with special arithmetic and number systems*, chapter Fault-tolerant computing in redundant residue number system, pages 65–88. Springer, 2017.
40. Taylor. Residue arithmetic a tutorial with examples. *Computer*, 17(5):50–62, 1984.
41. Sugandha Tiwari, Neel Gala, Chester Rebeiro, and V Kamakoti. PERI: A configurable posit enabled risc-v core. *ACM Transactions on Architecture and Code Optimization (TACO)*, 18(3):1–26, 2021.
42. Maria V Valueva, NN Nagornov, Pavel Alekseevich Lyakhov, Georgii V Valuev, and Nikolay I Chervyakov. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and computers in simulation*, 177:232–243, 2020.

43. Matthew Walker, Sascha Bischoff, Stephan Diestelhorst, Geoff Merrett, and Bashir Al-Hashimi. Hardware-validated CPU performance and energy modelling. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 44–53. IEEE, 2018.
44. Andrew Waterman and Krste Asanovi. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA version 20191213*. EECS Department, University of California, Berkeley, 2019.