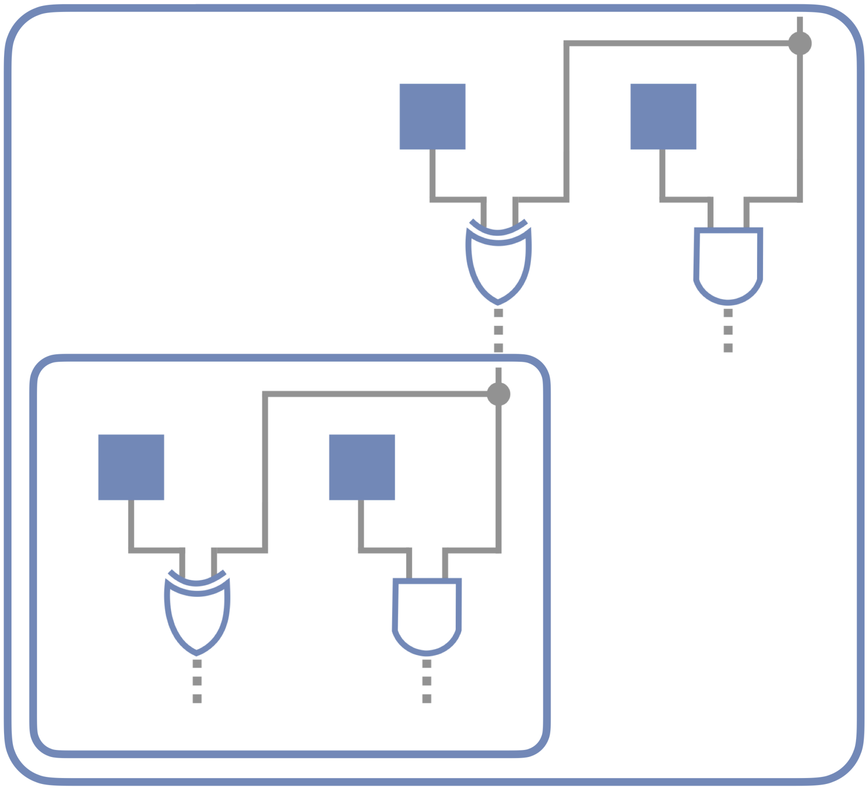
Inventions No. 3
sci–sci.org/2aalus
DOI: 10.5281/zenodo.17065633

SciSci Research
サイサイ・リサーチ

Future Computing



Hensel CPU Arithmetic Logic Units

Circuit Design for Exact Computing with 2-adic Arithmetic

James Douglas Boyd

SciSci Research, Inc. + Future Computing

SciSci Research, Inc.
Boulder, Colorado, United States
www.sci-sci.org

Citation Format:

Boyd, J.D. (2025). Hensel CPU Arithmetic Logic Units: Circuit Design for Exact Computing with 2-adic Arithmetic. SciSci Inventions, 1(3). DOI: 10.5281/zenodo.17065633

Contents

1 2-Adic Arithmetic Logic

The Hensel CPU is being developed by SciSci Research and Future Computing to realize an exact computing capability to replace floating-point computation. Unlike floating point numbers, which are approximations of numbers in $\mathbb{R}$, the Hensel CPU performs exact arithmetic in $\mathbb{Q}_2$, or, moreover, on finite encodings of 2-adic numbers. Arithmetic operations are executed in the Hensel CPU processor by 2-adic arithmetic logic units (2AALUs). At the circuit level, 2AALU operations on FC-3-2025-encoded operands (i.e., finite encodings of coefficients of 2-adic expansions) are performed via combinational logic by Boolean logic gates. Indeed, coefficient values in 2-adic expansions are either 0 or 1, as, in turn, are the entries in FC-3-2025 encodings of operands. Thus, exact computing in $\mathbb{Q}_2$ with the Hensel CPU is compatible with extant MOSFET technology; 2AALUs compute in bits with nothing so exotic as AND and XOR gates.

Previous reports on the Hensel CPU and Virtual Hensel I, written to introduce and demonstrate architecture and functionality (particularly regarding the processor), have done so at the expense of giving a circuit-level description of 2AALUs. Instead, for purposes of convenience, a description of 2AALU operations has been given in terms of "hop calculus", a higher-level description of 2AALU operations that abstracts away circuit-level logic. However, being given without reference to combinational logic, previous reports have failed to convey the simplicity and consistency with which 2AALU design accords with the rest of the Hensel CPU. Thus, it is necessary that a resource be provided on the fine details of 2AALU arithmetic; the following report is written for this purpose.

Necessarily building upon previous reports, the discussion herein offers recapitulation of hitherto-covered topics sparingly. Readers are encouraged to read previous reports as prerequisite material.

2 2AALUs and Hensel Processor Arithmetic

2.1 Circuit Design

The Hensel processor performs exact arithmetic in a largely parallelized fashion (plus a small $O(n)$ term). Processor combinational logic takes place in a circuit involving many 2AALUs. The circuit itself can, with some abstraction, be viewed as a tree whose vertices are 2AALUs.

The multi-level, nested design of the Hensel processor has been subject to extensive discussion in previous reports. Often, in discussion of the 2AALU cluster, one specifies its nest depth, $\mathcal{L}$. For instance, the Virtual Hensel I processor cluster is of nest depth $5\ (+2)$, i.e., it has five 2AALU levels and two additional levels (one for the clustered processor registers, $\mathcal{C}_{\text{PR}}$, at level $\ell = 1$ and one for the master processor register and master 2AALU, $\mathcal{M}_{\text{PR}}$ and $\mathcal{M}_{\text{AALU}}$, at level $\ell = \mathcal{L}$). At the circuit level, a processor with nest depth $\mathcal{L}$ will have a circuit tree with $2 + 2^{\mathcal{L}-2}$ vertices (with each tree being a DAG, $\mathcal{M}_{\text{PR}}$ being the source vertex, and the $\mathcal{C}_{\text{PR}}$ being the sink vertices.)

2AALUs perform arithmetic on operands according to FC-2-2025 instructions, which guide the 2AALUs in modifying the entries in FC-3-2025 operand encodings. 2AALUs perform modifications in a multi-level fashion, with a processor of nest depth $\mathcal{L} = n\ (+2)$ performing modifications on length-n operands, with each entry modified by a 2AALU at a different level. Thus, each entry is modified by a different vertex at a different level in the circuit tree. We write the levels in descending order, beginning with the source, $\mathcal{M}_{\text{PR}}$, at $\ell = \mathcal{L}$ and ending with the $\mathcal{C}_{\text{PR}}$ sinks at $\ell = 1$. In terms of level-wise modification, the leftmost entry in the operand is modified by a 2AALU at level $\ell = \mathcal{L} - 1$ and the rightmost entry is modified by a 2AALU at level $\ell = 2$.

2AALUs receive two kinds of inputs. Each circuit tree includes an initiator input Ψ, issued from level $\ell = \mathcal{L}$, which is always of value 1.

Each 2AALU also has its own modificatory input Φ, whose value depends on the FC-2-2025 instruction for the arithmetic at hand.

2.2 Combinational 2AALU Logic

The first 2AALU is found at level $\ell = \mathcal{L} - 1$. Ψ sends a 1 to both the XOR and AND gates of this $\ell = \mathcal{L} - 1$ 2AALU. The 2AALU gates also receive a Φ input. One of the gates, fed this input pair from Ψ and Φ, will give a 1 as output, the other 0. Each gate provides input for a 2AALU at the next level, giving a tree structure as illustrated in Figure 1. A 2AALU is thus a tuple $(\text{AND}, \text{XOR}, \Phi)$, and the circuit is a tree built from 2AALUs. The computation continues downstream of the gate that gives a 1 output, terminating at a $\mathcal{C}_{\text{PR}}$ at level $\ell = 1$. For each 2AALU, the computation involves no more than feeding the received input from the 2AALU at the previous tree level, as well as the Φ input, to its two logic gates.

Let's consider an example where all the Φ inputs are 0. Given a circuit of tree depth $\mathcal{L}$, the output will be of length $\mathcal{L} - 2$ with entries all being of value 1. Let's see why. Suppose the nest depth is $\mathcal{L} = 5$. Here, the output is $(1, 1, 1)$. The leftmost entry is determined at tree level $\ell = 4$, the second at tree level $\ell = 3$, and the third entry determined at tree level $\ell = 2$. Ψ sends a 1, as always. Φ_4 will, in this case, send a 0. So, the inputs at $\ell = 4$ are 1 and 0. The AND gate will produce a 0, and the XOR gate will produce a 1, since $1 \wedge 0 = 0$ and $1 \veebar 0 = 1$. Thus, the computation proceeds to the vertex downstream of the XOR gate. At the next level, $\ell = 3$, an AND gate and XOR gate are fed the 1 from the XOR gate at $\ell = 4$. We'll write these new gates as Γ_3^{AND} and Γ_3^{XOR}. So, Γ_3^{AND} and Γ_3^{XOR} receive a 1 from Γ_4^{XOR} and another input from Φ_3. In this case, Φ_3 will also give a 0 (since we're considering the example of all Φ values being 0). Thus, again, Γ_3^{XOR} will yield a 1, and Γ_3^{AND} will yield a 0. At $\ell = 2$, Γ_2^{AND} and Γ_2^{XOR} receive a 1 from Γ_3^{XOR} and a 0 from Φ_2, so Γ_2^{AND} will give a 0, and Γ_2^{XOR} will give a 1. Downstream of Γ_2^{XOR} is a $\mathcal{C}_{\text{PR}}$ whose address $\mathfrak{A}(\mathcal{C}_{\text{PR}})$ matches the output values in the tree: its address is $(1, 1, 1)$. Thus, with respect to the tree, we took a path of three consecutive XOR gates and terminated at the $(1, 1, 1)$-addressed processor register.

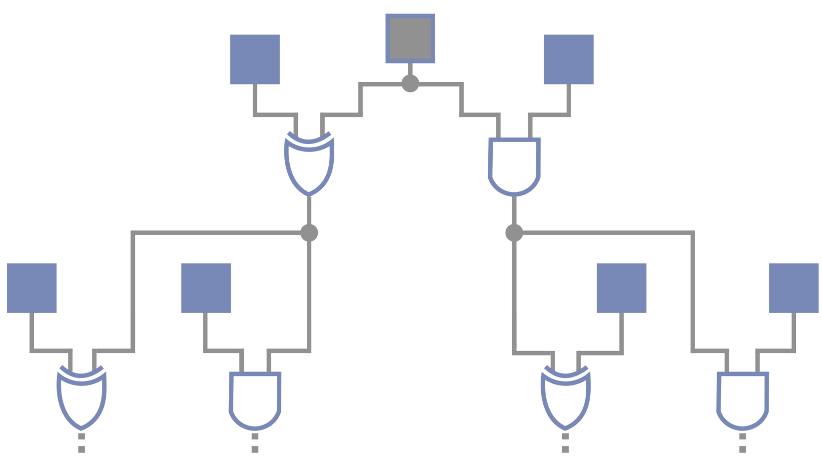


Figure 1: An illustration of a circuit tree with three 2AALUs. Each blue square is a Φ input, and the gray square is the Ψ input.

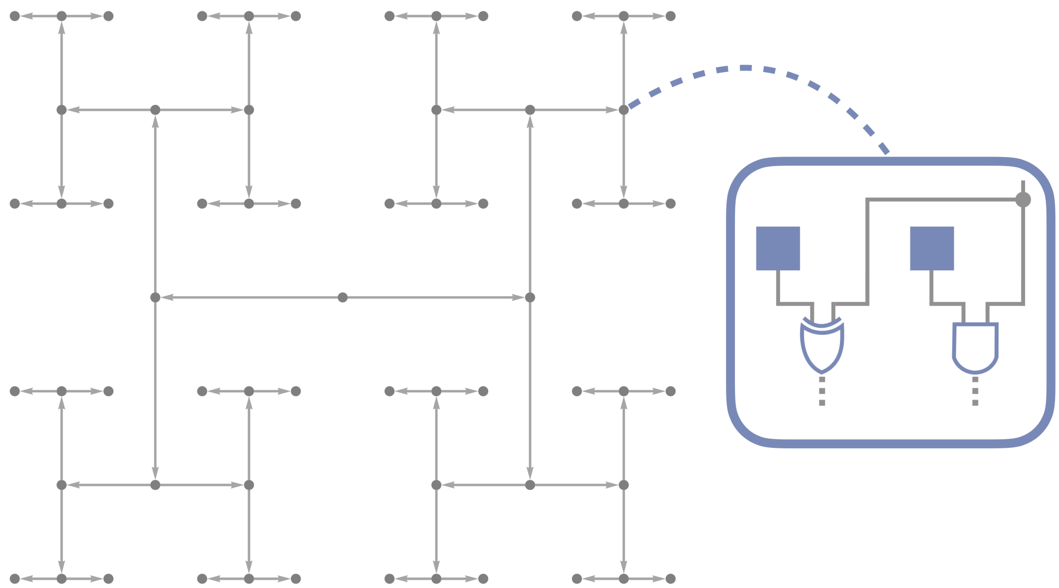


Figure 2: An illustration of the tree structure of the Hensel processor circuit, with each vertex its own 2AALU, as visualized via a callout.

Next, suppose we modify $(1,1,1)$ to $(1,0,1)$. The modification is given by Φ_3: it gives a 1. Thus, Γ_3^{XOR} will give an output of 0, and Γ_3^{AND} will give an output of 1; now we've taken a new path down the tree, downstream of Γ_3^{AND}.

The $\mathcal{C}_{\mathrm{PR}}$ are positioned at the ends of circuit tree paths such that, for a given path, its $\mathcal{C}_{\mathrm{PR}}$ has an address whose entries are the same as the XOR gate values computed by the 2AALUs that compose the path. In this case, the address is $(1,0,1)$, as were the XOR gate values. We can be more precise. Let $\mathfrak{C} = (\mathrm{V}, \mathrm{A})$ be a directed graph with vertices V and directed edges given by ordered pairs A, where $\deg^{+}(\mathrm{v_i} \in \mathrm{V}) = 2$ and $\deg^{-}(\mathrm{v_i} \in \mathrm{V}) = 1$ (for $\mathrm{i} \geq 1$), $\deg^{-}(\mathrm{v_i} \in \mathrm{V}) = 0$ for the case of $\mathrm{i} = 0$ (i.e., the source vertex), and $\deg^{+}(\mathrm{v_k} \in \mathrm{V}) = 0$ where the $\mathrm{v_k}$, given $|\mathrm{V}| = 2 + \sum_{\mathrm{n}=0}^{\mathrm{N}=2} 2^{\mathrm{n}}$, are the final $2^{\mathrm{N}-2}$ vertices (i.e., the sinks). A path in the circuit tree can be written as $\Pi = (\mathrm{V}', \mathrm{A}')$, where $\mathrm{V}' \subset \mathrm{V}$, $\mathrm{A}' \subset \mathrm{A}$, and $\mathrm{A}' = (\mathrm{v}'_1, \mathrm{v}'_2), (\mathrm{v}'_2, \mathrm{v}'_3), \ldots, (\mathrm{v}'_{\mathcal{L}-1}, \mathrm{v}'_{\mathcal{L}})$. Each v'_{i}, for $2 \leq \mathrm{i} \leq \mathcal{L} - 1$, corresponds to a 2AALU. Let ${}^{\mathrm{val}}\Gamma_{\ell}^{\mathrm{XOR}}(\mathrm{v}'_{\mathrm{i}})$ be the XOR-gate value of a given 2AALU. Then, in A′, the vertex $\mathrm{v}'_{\mathcal{L}}$ is a $\mathcal{C}_{\mathrm{PR}}$ whose address $\mathfrak{A}$ is $\left({}^{\mathrm{val}}\Gamma_{\ell}^{\mathrm{XOR}}(\mathrm{v}'_1), \ldots, {}^{\mathrm{val}}\Gamma_{\ell}^{\mathrm{XOR}}(\mathrm{v}'_{\mathcal{L}-1})\right)$. This is by design. With each path Π terminating in a different and unique $\mathcal{C}_{\mathrm{PR}}$, we can write the path as $\Pi\left(\mathcal{A}^{\left({}^{\mathrm{val}}\Gamma_{\ell}^{\mathrm{XOR}}(\mathrm{v}'_1), \ldots, {}^{\mathrm{val}}\Gamma_{\ell}^{\mathrm{XOR}}(\mathrm{v}'_{\mathcal{L}-1})\right)}\right)$. One thus gets χ-$\mathfrak{A}$ matching for free; χ-modification via Φ-perturbation directs the circuit tree path to the $\mathcal{C}_{\mathrm{PR}}$ with the matching address.

As another example, suppose we have a nest depth of $\mathcal{L} = 7$. Let our input operand be $(0,1,1,0)$, with our arithmetic to yield $(1,0,1,1)$. In this case, we begin with $\Pi\left(\mathfrak{A}^{(0,1,1,0)}\right)$ and obtain $\Pi\left(\mathfrak{A}^{(1,0,1,1)}\right)$. The changes are executed by Φ_5, Φ_4, and Φ_2. We will call the effectuated change-of-path under χ-modification a Φ-perturbation. That is to say, χ-modification is performed at each level, and the overall change effectuated in the circuit tree is a Φ-perturbation, with χ-modification giving a new operand and Φ-perturbation giving a new Π.

3 2AALUs and the CPU

3.1 2AALU Packaging and Nested Design

Naturally, a given path in the circuit tree will necessarily progress down multiple tree levels. Arithmetic operations amount to Φ_ℓ-perturbations applied at each level ℓ. The Hensel processor is itself of clustered form, designed to deliver Φ_ℓ-perturbations in parallel, by arranging 2AALUs at multiple levels. The multi-level design is modular and nested. A given 2AALU at level $\ell = n$ (where $2 \leq n \leq \mathcal{L} - 1$) is equipped with two logic gates, each of which, in turn, feeds into a 2AALU at level $\ell = n - 1$. Hardware-wise, each $\ell = n$ 2AALU is packaged in a carrier. In the case of the Hensel processor, these carriers are packaged inside one another, such that the carrier for a 2AALU at level $\ell = n - 1$ fed by a 2AALU at level $\ell = n$ is packaged within the carrier for the $\ell = n$ 2AALU, with the $\ell = n - 1$ carrier necessarily of smaller size than the $\ell = n$ 2AALU carrier. (It should be noted, however, that although being packaged within one another, all 2AALUs are nonetheless to be directly surface-mounted to the printed circuit board.) Nested packaging as such is illustrated in Figure 3.

The correspondence between carrier packaging and circuit tree structure is then as follows. A given 2AALU in a circuit tree will be packaged in the carrier for the 2AALU that is its parent vertex in the tree, as well as the other 2AALU which shares the same parent vertex. For a given 2AALU vertex $v \in V$ in $\mathfrak{C}$, let $\mathcal{K}$ be a set representing the carrier package for the 2AALU. The nested packaging procedure for the Hensel CPU processor is then as follows:

$$(v_i, v_j) \in A \implies \mathcal{K}(v_j) \subset \mathcal{K}(v_i) \tag{1}$$

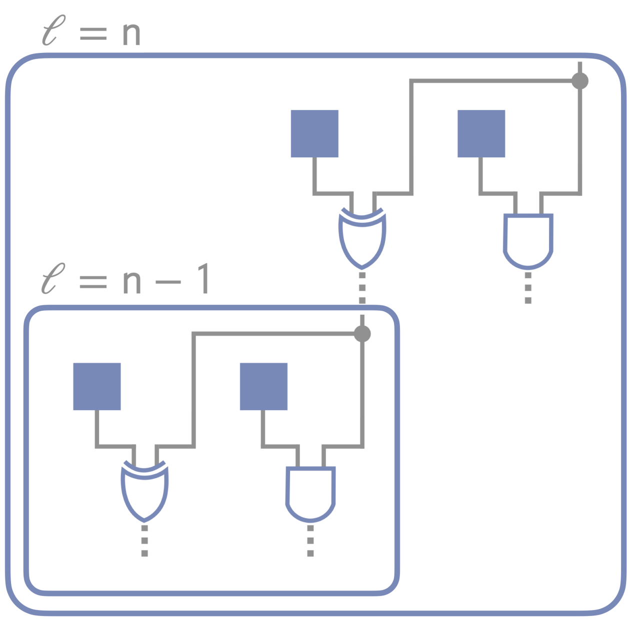


Figure 3: Illustration of nested packaging for 2AALU carriers.

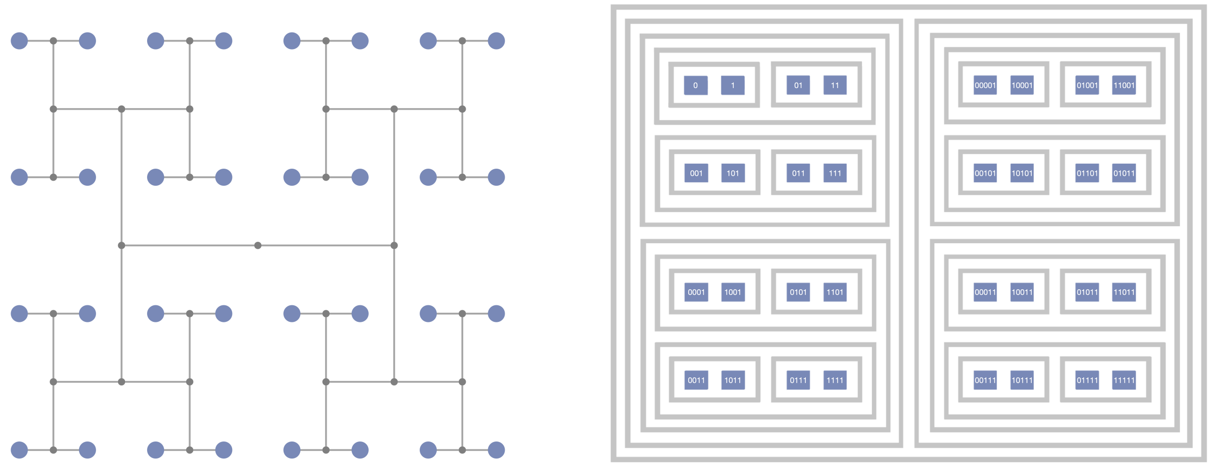


Figure 4: Comparison of 2AALU combinational logic tree structure and processor register distribution. The processor registers are highlighted blue (as sink vertices in the tree on the left and as occupants of level $\ell = \mathcal{L}$ in the nested processor structure on the right).

As illustrated in Figure 4, there is a correspondence between the tree structure of 2AALU combinational logic and the spatial distribution of processor registers in the Hensel processor cluster. This correspondence is the entirely by design: each path in the circuit tree terminates at a distinct processor register, and processor registers are positioned in the processor cluster according to their location in the circuit tree.

3.2 π-Sequence Passing

Following a computation, the new operand, the output, must in turn be taken up by the load-store architecture of the Hensel CPU, as described in previous reports. Not unlike the case of hop calculus, the term "π-sequence passing" has been used previously as a high-level shorthand for this process. Let us consider a more rarefied description herein. For expository gentleness, let us begin with the non-parallelized case where, at any given time, the master processor register, $\mathcal{M}_{\text{PR}}$, is loaded with a particular operand. The FC-3-2025 encoding of this operand is decompounded into χ-IDs, which are then loaded to cluster processor registers with matching addresses. Each $\mathcal{C}_{\text{PR}}$ has an addresses $\mathfrak{A}^{\chi}_{\pi}$, where χ is the matching ID and π is the π-sequence. $\mathcal{C}_{\text{PR}}$-loading is described via load operations λ. (See the Virtual Hensel report).

Once an arithmetic computation in the processor has been completed, the new $\mathcal{C}_{\text{PR}}$ at which the circuit tree path terminates under Φ-perturbation must in turn be loaded to the $\mathcal{M}_{\text{PR}}$ in place of the original input operand. This is achieved via "re-loads" λ_{re}:

$$\lambda^{\text{A}}_{\text{re}} : \mathfrak{A}^{\chi}_{(*,-,-)} \to \mathfrak{A}^{\mu(\chi)}_{(*,-,-)} \tag{2}$$

$$\lambda^{\text{B}}_{\text{re}} : \mathfrak{A}^{\chi}_{(-,*,-)} \to \mathfrak{A}^{\mu(\chi)}_{(-,*,-)} \tag{3}$$

$$\lambda^{\text{C}}_{\text{re}} : \mathfrak{A}^{\chi}_{(-,-,*)} \to \mathfrak{A}^{\mu(\chi)}_{(-,-,*)} \tag{4}$$

$$\lambda^{\text{D}}_{\text{re}} : \mathfrak{A}^{\chi}_{(-,*,-)} \to \mathfrak{A}^{\mu(\chi)}_{(-,*,-)} \tag{5}$$

where μ is a modification, and $\mu(\chi)$ being the χ-modified ID encoding the output operand. (The significance of the subscripts, such as $(*,-,-)$ pertains to the R_{FC}, L_{FC}, and T_{FC} blocks in the FC-3-2025 encoding of the operand, i.e., its χ^{∂}-IDs, as described in the Virtual Hensel Report. A discussion of loads, i.e., λ mappings, can be found in the same report.)

Re-loads give a precise description of what in previous reports is described via the shorthand of "π-sequence passing": the π-sequences that were originally λ-loaded to $\mathcal{C}_{\mathsf{PR}}$ whose addresses matched the input χ^{∂}-IDs are now re-loaded (i.e., λ_{re}-loaded) to the output-χ^{∂}-matching $\mathcal{C}_{\mathsf{PR}}$, such that the output χ^{∂}-IDs, rather than the input χ^{∂}-IDs, are now loaded to $\mathcal{M}_{\mathsf{PR}}$. Following these re-loading operations, the output χ^{∂}-IDs in $\mathcal{M}_{\mathsf{PR}}$ are recompounded and sent to the load-store unit (as discussed in the Virtual Hensel report).

3.3 Hop Calculus

As discussed, the arithmetic operations themselves are manifest in the Φ_ℓ inputs. The processor executes all Φ_ℓ in parallel. Then, the processor runs the path computation of the Φ-perturbation, which is not parallel, but imposes but a small additional linear term, $O(n)$, where $n = \mathcal{L} - 1$. These Φ_ℓ are the so-called "hop operations" in hop calculus; Φ_ℓ inputs cause the path to "hop" from the input-terminating path to the output-terminating path. One gets a new Π by changing at least one Φ_ℓ value; thus, one can describe 2AALU arithmetic in terms of Φ inputs alone, i.e., in terms of hops, sparing the details of its underlying combinational logic. Thus, it is convenient to describe an arithmetic operation in hop calculus. For instance, returning to the previous example of the computation yielding $\Pi\left(\mathfrak{A}^{(1,0,1,1)}\right)$ from $\Pi\left(\mathfrak{A}^{(0,1,1,0)}\right)$, the Φ_ℓ-perturbations can be described solely in terms of hops: $\left(h_5^{\sigma}, h_4^{\overline{\sigma}}, h_2^{\sigma}\right)$. Here, a h_ℓ^{σ} hop is a Φ_ℓ input giving a 1 and a $h_\ell^{\overline{\sigma}}$ hop is a Φ_ℓ input giving a 0. (See the Virtual Hensel report for a review of hop calculus notation.)

We can more precisely describe the relationships between hops and Φ_ℓ inputs. Suppose we have a χ^{∂}-ID loaded to a given cluster processor register. Let $\chi^{\partial}_{(-,-,*)} = (1,1,1,0,1)$, which is the T_{FC} block for the FC-3-2025 encoding of $\frac{29}{32}$. Suppose our arithmetic operation modifies this χ^{∂} block to $(1,0,0,0,0)$. (For instance, if the arithmetic is $\frac{29}{32} - \frac{13}{32}$, the output encoding will include the single block $\chi^{\partial}_{(-,-,*)} = (1,0,0,0,0)$.) Looking at the IDs, one sees that three modifications are in order, namely at Φ_6, Φ_4, and Φ_3. Figure 5 illustrates both tree paths, with the Π segments that differ by Φ_ℓ input highlighted purple.

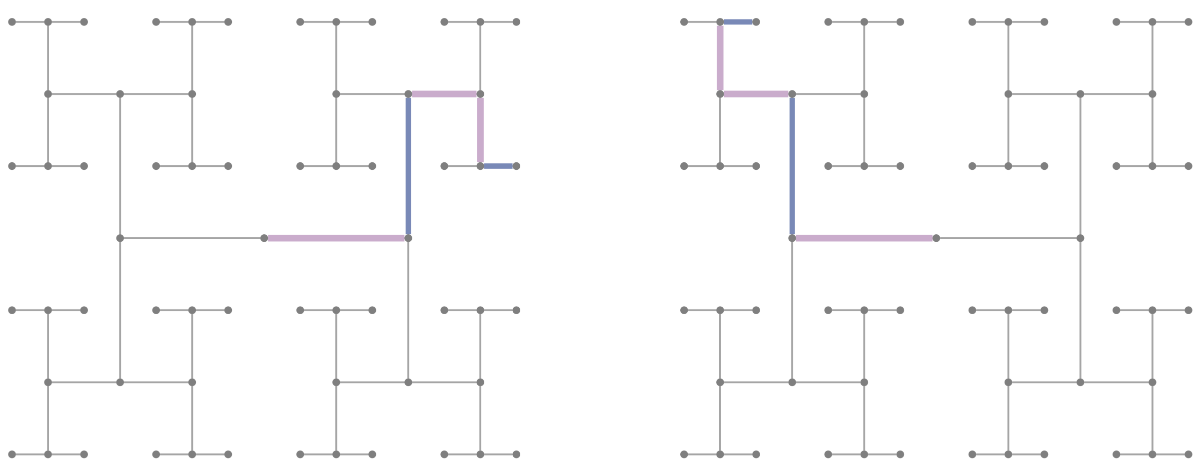


Figure 5: An illustration of $\Pi\left(\mathfrak{A}^{(1,1,1,0,1)}_{-,-,*}\right)$ and $\Pi\left(\mathfrak{A}^{(1,0,0,0,0)}_{-,-,*}\right)$. Φ_6, Φ_4, Φ_3 are highlighted purple.

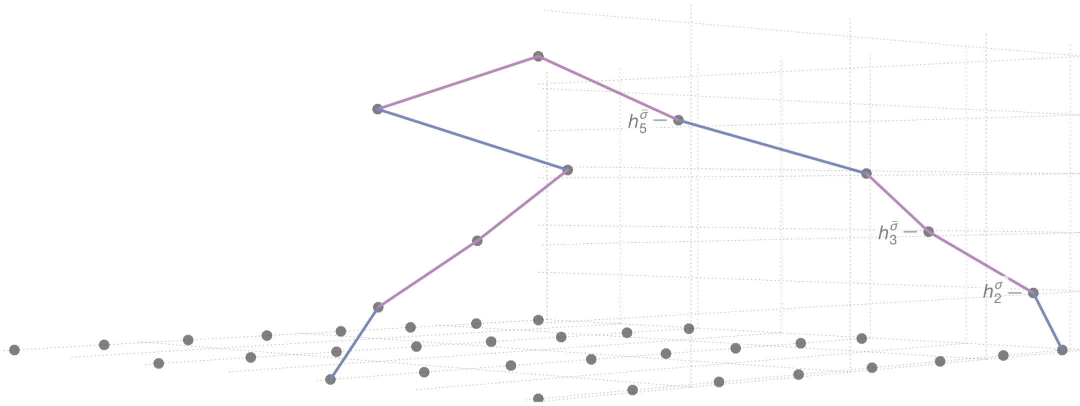


Figure 6: A three-dimensional illustration of Figure 5, with ℓ levels distinguished along the z axis.

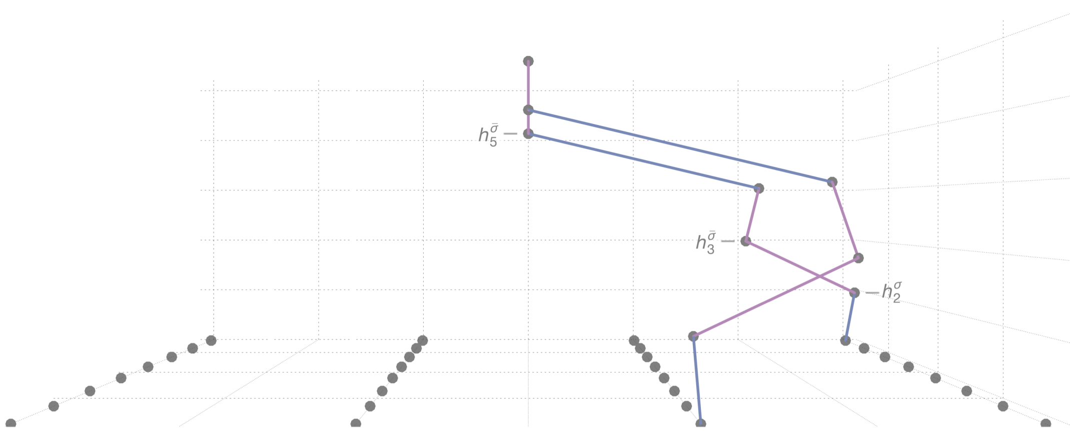


Figure 7: Figure 5 as seen from a different angle.

As one can see, Φ_6, Φ_4, and Φ_3 divert the tree path from the processor register with address $\mathfrak{A}^{(1,1,1,0,1)}$ to that with address $\mathfrak{A}^{(1,0,0,0,0)}$. Φ_6 directs the tree leftward, rather than rightward; Φ_4 also directs the tree leftward rather than rightward; and Φ_3 directs the tree upward rather than downward. Figures 6 and 7 give three-dimensional illustrations, with ℓ-level parameterized by the z-axis, and the points at level $\ell = 1$ being the $\mathcal{C}_{\text{PR}}$ at which the computations terminate. These figures illustrate the effect of Φ-perturbations on both circuit tree paths and the processor register destinations.

3.4 Parallelization and Scaling

Thus far, our descriptions have been restricted to a Hensel processor with a single circuit tree. However, there is no reason why multiple trees cannot be employed, so long as each $\mathcal{C}_{\text{PR}}$ is designed to accept multiple inputs and the 2AALU carriers are designed to package multiple trees. Multiple trees would allow for parallelization of FC-2-2025 instruction execution; the processor could perform arithmetic on multiple inputs via multiple circuit trees.

SciSci Research

サイサイ・リサーチ

Published by SciSci Press