A Mixed-Radix Positional Notation with Per-Chunk Bases

Abstract
We introduce a generalized positional number system in which each digit-block ("chunk") may use its own base. In particular, we extend the standard decimal notation (single base 10) to a "multi-dot" notation where each segment between decimal points is interpreted in an arbitrary radix bi. We formally define the **Mixed-Radix Block Notation** (MRBN), demonstrate how it recovers familiar systems (e.g., time hh:mm:ss, currency pounds–shillings–pence, and "base-100" chunking of ordinary decimals), and prove that any MRBN representation maps unambiguously to a unique integer or real number. We highlight applications in human-readable timestamps, hierarchical currency, embedded-system data-packing, and pedagogical tools. Finally, we list avenues for further theoretical investigation, propose a set of formal Patent Claims, and discuss how MRBN may be extended or combined with conventional floating-point.
Keywords: mixed radix, positional notation, base-100, hierarchical units, fixed-point representation, embedded data formats, numerical systems, invention disclosure.

1. Introduction

Traditional positional numeral systems assign a single base B to every digit position. For instance, in base-10 (decimal), a number

$$D_n D_{n-1} \dots D_1 D_0 . D_{-1} D_{-2} \dots D_{-m}$$

is interpreted as

$$\sum_{i=-m}^{n} D_i B^i, \quad B = 10, \quad 0 \le D_i < B.$$

By contrast, many real-world domains (time, currency, measurement units, sensor encodings) naturally use **mixed-radix** representations, e.g., hours:minutes:seconds (base 24 : 60 : 60) or pounds:shillings:pence (base 20 : 12 : 1). While mixed-radix integers and time arithmetic have long been standard, we here propose a **general-purpose positional system**—termed the **Mixed-Radix Block Notation** (MRBN)—in which each "digit chunk" is separated by a dot and may employ a distinct radix b_i, with clear and unambiguous conversion to and from standard integers or reals. When each $b_i = 100$, one recovers the "base-100 block" reinterpretation of ordinary decimals (e.g., "99.90.9.01"-> 99.90901 base 10).

Our contributions are:

1. A rigorous formal definition of MRBN (integer and fractional variants).
2. Proof of uniqueness (one-to-one mapping between MRBN strings and numerical values), existence of greedy conversion algorithms, and basic arithmetic properties (addition, carry rules).
3. Examples of special cases: single-base (ordinary decimal, base 100), standard mixed-radix (time, currency), and novel chunkings for high-precision human-readable timestamps (sub-unit subdivisions beyond two decimal places).
4. Analysis of storage/computational advantages in embedded and BCD-like contexts.
5. A "claims" section suitable for invention patent disclosure, covering the core MRBN concept and key applications.

The remainder of this paper is organized as follows. In Section 2, we review related notational systems and motivate MRBN. In Section 3, we formalize the definition of MRBN (integer and fractional forms) and prove core properties (uniqueness, validity). Section 4 presents detailed examples and algorithms for conversion to/from standard integer/floating-point. Section 5 surveys potential applications (timestamps, currency, data-packing, pedagogy). In Section 6, we discuss advantages, limitations, and directions for future work. Section 7 concludes. Finally, Section 8 states formal Patent Claims.

2. Background and Related Work

2.1 Positional Systems and Fixed Base

A **positional numeral system** over a single radix B expresses any nonnegative integer N uniquely as

$$N = \sum_{i=0}^{n} d_i\, B^i, \quad 0 \le d_i < B,\ d_n \ne 0.$$

Similarly, a nonnegative real number x is written

$$x = \sum_{i=-m}^{n} d_i\, B^i, \quad 0 \le d_i < B.$$

When $B = 10$, we call it **decimal**; when $B = 2$, **binary**; and so forth. In higher-precision contexts (IEEE 754 floating-point), one typically fixes $B = 2$ or $B = 10$, a fixed exponent range E, and a fixed significand precision.

2.2 Mixed-Radix Systems

A **mixed-radix integer system** allows each digit position to have its own radix b_i. That is, given a finite tuple $(b_n, b_{n-1}, \dots, b_0)$, each tuple of digits $(d_n, \dots, d_0)$ with $0 \le d_i < b_i$ denotes the integer

$$N \; = \; d_0 + d_1\, b_0 + d_2\,(b_0\, b_1) + \cdots + d_n\,(b_0\, b_1\, \cdots\, b_{n-1}).$$

Common real-world examples include:

- **Time notation**: hours – minutes – seconds : $b_2 = 24, b_1 = 60, b_0 = 60.$
- **Currency (pre-decimal UK)**: pounds – shillings – pence : $b_2 = \infty$ (unbounded)$, b_1 = 20, b_0 = 12.$
- **Angle measure**: degrees – arcminutes – arcseconds : $b_2 = \infty, b_1 = 60, b_0 = 60.$

Mixed-radix integer encoding ensures **unique representation** when each digit d_i satisfies $0 \le d_i < b_i$.

2.3 Fixed-Point Representations and BCD

In many embedded systems, one uses **fixed-point** or **Binary-Coded Decimal (BCD)** formats. A typical BCD scheme packs two decimal digits (00..99) into one byte (base 100). This reduces ASCII overhead for decimal digits and simplifies software logic by grouping pairs of decimal digits. Our proposed MRBN with all bi=100 recovers a generalization of base-100 BCD to an arbitrary number of decimal places.

3. Formal Definition of Mixed-Radix Block Notation (MRBN)

3.1 Notation and Preliminaries

- Let $\mathbf{b} = (b_0, b_1, \dots, b_n)$ be a finite sequence of integers with each $b_i \ge 2$.
- Let $\mathbf{D} = (D_n, D_{n-1}, \dots, D_0)$ be a corresponding **digit sequence**, where each D_i is an integer satisfying

$$0 \; \le \; D_i \; < \; b_i.$$

- We write the **Mixed-Radix Block Notation** (MRBN) for $(D_n, \dots, D_0)$ as

$$D_n \,.\, D_{n-1} \,.\, \dots \,.\, D_0.$$

  In the **pure integer form**, this represents

$$\text{Value}(D_n. \dots .D_0) \; = \; \sum_{i=0}^{n} D_i \,\Big(\prod_{j=0}^{i-1} b_j\Big), \quad \prod_{j=0}^{-1} b_j := 1.$$

Concretely:

- The rightmost block D_0 has weight $w_0 = 1$.
- The next block D_1 has weight $w_1 = b_0$.
- In general,

$$w_i = \prod_{j=0}^{i-1} b_j, \quad 0 \le i \le n, \quad (\text{empty product} = 1).$$

- Hence,

$$D_n.D_{n-1}.\cdots.D_0 \;=\; D_n\,(b_{n-1}\,b_{n-2}\cdots b_0) \;+\; D_{n-1}\,(b_{n-2}\cdots b_0) \;+\;\cdots\;+\; D_1\,b_0 \;+\; D_0.$$

3.2 Fractional (Real-Valued) MRBN

To encode a nonnegative real number with fractional part, we extend the digit-block sequence to the left of a "decimal point" and allow negative indices. Let

$$\mathbf{b} \;=\; (\ldots, b_3, b_2, b_1, b_0, b_{-1}, b_{-2}, \ldots, b_{-m}),$$

and let

$$\mathbf{D} \;=\; (\ldots, D_3, D_2, D_1, D_0.D_{-1}, D_{-2}, \ldots, D_{-m}),$$

with each D_i satisfying $0 \le D_i < b_i$. Then define the **place-value weights**:

- For $i \ge 0$:

$$w_i = \prod_{j=0}^{i-1} b_j, \quad w_0 = 1.$$

- For $i < 0$:

$$w_i = \prod_{j=i}^{-1} b_j^{-1} \;=\; \frac{1}{b_i\,b_{i+1}\,\cdots\,b_{-1}}.$$

Hence the **numerical value** of

$$\dots D_2\ D_1\ D_0.D_{-1}\ D_{-2}\ \dots\ D_{-m}$$

is

$$\underbrace{\sum_{i=0}^{\infty} D_i\, w_i}_{\text{integer part}} + \underbrace{\sum_{i=-1}^{-m} D_i\, w_i}_{\text{fractional part}}.$$

In practice we fix a finite range $-m \le i \le n$ so only $n+1$ integer-blocks and m fractional-blocks appear.

3.2.1 Uniqueness of Representation

Theorem 3.1 (Uniqueness)

Given a finite block of bases $\{b_{-m}, \dots, b_{-1}, b_0, \dots, b_n\}$ with each $b_i \ge 2$, any real number x in the interval

$$0 \;\le\; x \;<\; \sum_{i=0}^{\infty}(b_i - 1)\, w_i \;+\; \sum_{i=-1}^{-m}(b_i - 1)\, w_i$$

admits at **most one** MRBN representation

$$x = \sum_{i=0}^{n} D_i\, w_i \;+\; \sum_{i=-1}^{-m} D_i\, w_i, \quad 0 \le D_i < b_i,\ D_i \in \mathbb{Z}.$$

Proof sketch. The proof parallels standard radix-B uniqueness arguments: "Greedy" digit-selection at each place ensures maximal D_i such that $\sum_{j=i}^{-1} D_j\, w_j \le x$. More formally, assume two distinct digit-sequences $\{D_i\}$ and $\{D_i'\}$ produce the same real x. Let k be the most significant index where $D_k \ne D_k'$. Without loss, assume $D_k < D_k'$. Then

$$\sum_{i \ge k} D_i\, w_i < \sum_{i \ge k} D_i'\, w_i' \le \sum_{i \ge k}(b_i - 1)\, w_i \le \sum_{i > k}(b_i - 1)\, w_i + (D_k' + 1)\, w_k \le \cdots,$$

leading to a strict inequality versus the other representation of x, a contradiction. ∎

3.2.2 Existence and Greedy Algorithm

Given a target real x with $0 \le x < \sum_{i \ge 0}(b_i - 1)\, w_i + \sum_{i<0}(b_i - 1)\, w_i$, we can compute $\{D_i\}$ from high index $i = n$ down to $i = -m$ by:

1. Set $r := x$.
2. For $i = n, n-1, \dots, 0$, choose

$$D_i := \lfloor r/w_i \rfloor, \quad 0 \le D_i < b_i, \quad r := r - D_i\, w_i.$$

3. For the fractional part, for $i = -1, -2, \dots, -m$, choose

$$D_i := \lfloor r/w_i \rfloor, \quad 0 \le D_i < b_i, \quad r := r - D_i\, w_i.$$

Because w_i decreases by a factor b_i each time, one ensures $D_i < b_i$. This greedy selection yields the unique MRBN digits.

3.2.3 Addition and Carry Rules

To add two MRBN-encoded values $\{D_i\}$ and $\{E_i\}$ sharing the same per-index bases $\{b_i\}$, perform:

1. For each $i \in \{-m, \dots, n\}$, compute $S_i = D_i + E_i + C_i$ (where C_i is carry from previous "lower" index; initially $C_{-m} = 0$).
2. Set new digit $F_i = S_i \bmod b_i$.
3. Compute carry $C_{i+1} = \lfloor S_i / b_i \rfloor$.
4. Continue until $i = n$. If $C_{n+1} > 0$, overflow occurs (need extra chunk $D_{n+1} < b_{n+1}$ to hold it).
   This per-index carry rule is the hallmark of mixed-radix arithmetic: each column has modulus b_i.

4. Special Cases and Examples

4.1 Ordinary Single-Base Representation

If $b_i = B$ for all integer-index $i \ge 0$ and fractional-index $i < 0$, then MRBN exactly specializes to **base-B** positional notation.

In particular, setting all $b_i = 10$ recovers ordinary decimal; all $b_i = 2$ recovers binary; all $b_i = 16$ recovers hexadecimal.

4.2 Base-100 Block Notation (All bi=100)

Take $\{b_n, \dots, b_{-m}\} = (100,\ 100, \dots, 100)$. Then any MRBN string

$$D_n.D_{n-1}.\cdots.D_1.D_0.D_{-1}.\dots.D_{-m}$$

represents (using weights $w_i = 100^i$ for $i \geq 0$ and $w_i = 100^{\,i}$ for $i < 0$)

$$\sum_{i=-m}^{n} D_i\, 100^i.$$

Concretely, "99.90.9.01" (with only four blocks: $D_3 = 99,\ D_2 = 90,\ D_1 = 9,\ D_0 = 1$, and no negative-index fractional blocks) is interpreted as

$$99 \times 100^3 + 90 \times 100^2 + 9 \times 100^1 + 1 \times 100^0 \ = \ 99{,}900{,}901 \text{ (integer form)},$$

or, if one inserts a "decimal point" before D_0 to indicate fractional base-100,

$$99.90.9.01 \ = \ 99 \times 100^0 \ + \ 90 \times 100^{-1} \ + \ 9 \times 100^{-2} \ + \ 1 \times 100^{-3} \ = \ 99.900901_{10}.$$

Hence, any ordinary decimal with an even number of digits to the right of the decimal point can be grouped in two-digit blocks and written as an MRBN string in base 100.

4.3 Standard Mixed-Radix: Time Notation

Consider $b_2 = 24,\ b_1 = 60,\ b_0 = 60$. Then a time "$H : M : S$" with

$$0 \leq S < 60, \quad 0 \leq M < 60, \quad 0 \leq H < 24$$

denotes the integer (total seconds since midnight)

$$H \times (60 \times 60) \ + \ M \times (60) \ + \ S.$$

For fractional seconds (e.g., milliseconds, microseconds), we extend with $b_{-1} = 1000$ for milliseconds, $b_{-2} = 1000$ for microseconds, etc. A full timestamp like

$$13.45.37.123.456 \quad (13\text{h} : 45\text{m} : 37\text{s}.123\text{ms}.456\mu\text{s})$$

represents

$$13 \times (60 \times 60) + 45 \times 60 + 37 + \frac{123}{1000} + \frac{456}{1000^2} = 49537.123456 \text{ seconds after midnight.}$$

4.4 Currency: Rupees-Paise-Paisa

Take $b_2 = \infty$ (unbounded, for rupees), $b_1 = 100$ (paise in a rupee), $b_0 = 100$ (paisa in a paise). If one writes

$$15.45.30 \quad (15 \text{ rupees, } 45 \text{ paise, } 30 \text{ paisa})$$

this denotes

$$15 \times (100 \times 100) + 45 \times (100) + 30 = 15 \cdot 10{,}000 + 45 \cdot 100 + 30 = 154530$$

"paisa-units," i.e.\ ₹1545.30. Inverse conversion uses division by 100 repeatedly:

$$\text{Unit-count: } 154530 \xrightarrow{\div 100} (q = 1545,\ r = 30) \xrightarrow{\div 100} (q = 15,\ r = 45).$$

4.5 Arbitrary Mixed Bases per Chunk

More generally, let

$$(b_n, b_{n-1}, \ldots, b_1, b_0, b_{-1}, \ldots, b_{-m}) = (\beta_0, \beta_1, \ldots, \beta_{n+m})$$

where each $\beta_i \geq 2$. A digit-string

$$D_n.D_{n-1}.\cdots.D_1.D_0.D_{-1}.\cdots.D_{-m}$$

with $0 \leq D_i < b_i$ represents

$$\sum_{i=0}^{n} D_i \left(\prod_{j=0}^{i-1} b_j\right) + \sum_{i=-1}^{-m} D_i \left(\prod_{j=i}^{-1} b_j\right)^{-1}.$$

For example, if $\mathbf{b} = (60, 24,)$ for integer part only, we recover "$H : M$" in hours and minutes. If we choose

$$(b_2, b_1, b_0, b_{-1}, b_{-2}, b_{-3}) = (2,\ 3,\ 7,\ 12,\ 100,\ 100),$$

then "$D_2.D_1.D_0.D_{-1}.D_{-2}.D_{-3}$" is some exotic measure that uses binary, ternary, septenary, duodecimal, and base-100 subunits. Though contrived, it remains **unambiguous** as soon as the reader knows $\mathbf{b}$.

5. Applications and Use Cases

5.1 Human-Readable Timestamps with Sub-Millisecond Precision

Modern distributed systems often require **sub-millisecond logging** (nanosecond or picosecond resolution). Ordinarily, one might record time in pure integer nanoseconds T ns since epoch, then let the reader convert to HH:MM:SS.mmmuuu… However, MRBN allows direct human-readable notation such as:-

07 . 54 . 32 . 123 . 456 . 789

where:

- $b_5 = 24$ (hours),
- $b_4 = 60$ (minutes),
- $b_3 = 60$ (seconds),
- $b_2 = 1000$ (milliseconds),
- $b_1 = 1000$ (microseconds),
- $b_0 = 1000$ (nanoseconds).

If a log entry reads "07.54.32.123.456.789," one immediately knows it is 07 h 54 m 32 s 123 ms 456 µs 789 ns. Converting to a single integer of nanoseconds is then a matter of

$$7 \cdot (60 \cdot 60 \cdot 10^9) + 54 \cdot (60 \cdot 10^9) + 32 \cdot 10^9 + 123 \cdot 10^6 + 456 \cdot 10^3 + 789.$$

Because each block's base matches the natural rollover (24 h, 60 m, 60 s, 1000 ms, 1000 µs, 1000 ns), MRBN provides a **lossless**, **readable**, and **precise** multi-unit timestamp with minimal cognitive overhead.

5.2 Currency with Multiple Subunits

In certain regions, one rupee (₹) equals 100 paise, and one paise further subdivides into 100 "micro-paise" (hypothetical). An MRBN representation

$$R.P.\mu P,$$

with $(b_2 = \infty, b_1 = 100, b_0 = 100)$, lets merchants write "12.45.30" to mean

$$12 \text{ rupees} + 45 \text{ paise}/100 + 30 \text{ µP}/(100^2) = 12.4530₹.$$

Large commerce applications or point-of-sale terminals can store each chunk as a single byte:

- Byte #2: $0 \le R < 2^8$,
- Byte #1: $0 \le P < 100$,
- Byte #0: $0 \le \mu P < 100$.
  To compute total micro-paise (the smallest unit), one does

$$\text{Total} \;=\; R \times (100 \times 100) + P \times (100) + \mu P.$$

When receiving a payment of "07.12.99," the POS can validate instantly that $P < 100$ and $\mu P < 100$, then apply consistent carry-logic if needed (e.g.\ add two prices "07.12.99 + 03.89.54" and so forth).

5.3 Embedded-System Data Packing (Base-100 Bytes)

In low-power microcontrollers without floating-point units, **packed BCD** often encodes two decimal digits (00–99) per byte. MRBN generalizes this: if a 32-bit sensor reading naturally breaks into four two-digit chunks (each 0–99), one can store them as four bytes. Example:

$$\underbrace{D_3}_{0..99} \,.\, \underbrace{D_2}_{0..99} \,.\, \underbrace{D_1}_{0..99} \,.\, \underbrace{D_0}_{0..99}$$

Each D_i fits in one byte. The overall 32-bit integer is simply

$$N = D_3 \times (100^3) \;+\; D_2 \times (100^2) \;+\; D_1 \times (100^1) \;+\; D_0.$$

When transmitting over a low-bandwidth link, the receiver knows “we’re base-100, four bytes,” so no further translation is needed to recover the true reading. Compared to ASCII “0–9” codes (one digit per byte), MRBN here is **50 % more compact** for high-precision decimal-based sensor data.

5.4 Pedagogical Illustration of Positional Systems

We have observed that instructors of elementary arithmetic often teach base-2 (binary) and base-16 (hex) alongside base-10 to illustrate that “digit alphabets” need not be {0,…,9}. MRBN further extends this by allowing each position its own base, demonstrating the general concept:

> **Definition (Mixed-Radix Pedagogy).** A student writes “$d_3.d_2.d_1.d_0$” and is told “digit d_0 counts multiples of 1, d_1 counts multiples of b_0, d_2 counts multiples of $b_0 b_1$, etc.” By varying $\{b_i\}$, one can show, e.g.:
>
> - ““13 . 05 . 28” in $(b_2, b_1, b_0) = (60, 60, 24)$ is a time (13 h 05 m 28 s)”
> - ““4 . 17 . 03” in $(b_2, b_1, b_0) = (10, 100, 100)$ is “4 units + 17/100 + 03/10 000 = 4.1703.””
> - ““2 . 11 . 09” in $(b_2, b_1, b_0) = (3, 7, 12)$ is the integer $2 \cdot (7 \cdot 12) + 11 \cdot (12) + 9 = 2 \cdot 84 + 11 \cdot 12 + 9 = 168 + 132 + 9 = 309$.”

By exploring different $\{b_i\}$ choices, students see how radices determine the weight of each digit. MRBN thus provides a **concrete, hands-on** teaching aid bridging abstract “base” ideas with familiar concrete units (time, money, etc.).

6. Advantages and Future Directions

6.1 Advantages

1. **Exactness and Unambiguity**: Once the per-index base sequence $\{b_i\}$ is specified, every MRBN string corresponds to a unique integer or real number, and vice versa (Theorem 3.1).
2. **Natural Fit for Heterogeneous Units**: Domains that already use multi-tier units (e.g.\ 24 h/60 m/60 s, currency with coins and subcoins) can adopt MRBN directly, avoiding post-processing.
3. **Compactness in Fixed-Width Encodings**: When each base $b_i \leq 256$, each MRBN digit block can map to one byte. For example, base-100 (00–99) fits in one byte, packing two decimal digits into one byte (50 % space savings versus ASCII).
4. **Pedagogical Clarity**: Working with differing bases per digit deepens understanding of positional notation and place-value concepts.
5. **Extensibility**: MRBN extends immediately to rational and real numbers by allowing negative indices and fractional digits $\{D_{-1}, D_{-2}, \ldots\}$. One can even combine integer multipliers (e.g.\ primes) with fractional bases to create totally custom numbering schemes.

6.2 Future Directions

1. **Efficient MRBN Arithmetic**: Develop algorithms and hardware architectures that operate natively on mixed-radix representations—e.g.\ digit-serial mixed-radix adders and multipliers, analogous to binary/decimal ALUs.
2. **Error-Detection and Correction**: Explore checksums or parity schemes tailored to MRBN—e.g.\ "each chunk yields a check-digit mod b_i."
3. **Compression Schemes**: Combine MRBN with entropy coding (Huffman, arithmetic coding) to compress decimal-like data in a variable-length format, using chunk boundaries as natural codeword separators.
4. **Applications in Blockchains and Financial Ledgers**: Use MRBN to store multi-denominational cryptocurrencies (e.g.\ storing "coins.satoshis.microSatoshis" in one 24-byte structure).
5. **Formal Languages and Syntax**: Define context-free grammars or parsing algorithms for MRBN literals in programming languages. Propose extensions to JSON, YAML, or SQL for "multi-dot numeric literals."