A Decimal–Hexadecimal Block Encoding for Prime Storage with Modular Compression via the Ternary Constraint

Ricardo Adonis Caraccioli Abrego *

Posted Date: 9 April 2026

doi: 10.20944/preprints202604.0695.v1

Keywords: prime encoding; primality storage; prime database; prime-counting function; decimal block encoding; hexadecimal encoding; nibble encoding; modular compression; ternary constraint; residue classes modulo 10; residue classes modulo 3; lossless compression; Shannon entropy; popcount; wheel-based storage

 doi:10.20944/preprints202604.0695.v1

Article

A Decimal–Hexadecimal Block Encoding for Prime Storage with Modular Compression via the Ternary Constraint

Ricardo Adonis Caraccioli Abrego

Universidad Nacional Autónoma de Honduras (UNAH) Departamento de Ingeniería Eléctrica, Campus Cortés, Honduras; ricardo.caraccioli@unah.edu.hn

* Correspondence:

Abstract

We describe a decimal–hexadecimal block encoding for primality over a finite stored range. Since every prime greater than 5 must lie in one of the residue classes $1, 3, 7, 9 \pmod{10}$, each decimal block of size ten can be encoded by a 4-bit word indicating which of the candidates $10k+1$, $10k+3$, $10k+7$, and $10k+9$ are prime. This yields a nibble-based storage scheme supporting exact primality queries and exact recovery of the prime-counting function $\pi(x)$ by cumulative popcount. We then establish a structural theorem arising from the congruence $10 \equiv 1 \pmod{3}$: for $k \equiv 0 \pmod{3}$ the candidates $10k+3$ and $10k+9$ are always composite, and for $k \equiv 2 \pmod{3}$ the candidates $10k+1$ and $10k+7$ are always composite. This partitions the nibble alphabet into three classes of sizes 4, 16, and 4, reducing the Shannon entropy from 4 bits to 2.42 bits per nibble and yielding a lossless compression of 39.4% over the original encoding with $O(1)$ decode complexity. We present data structures, Python routines, and experimental validation up to 300,000.

Keywords: prime encoding; primality storage; prime database; prime-counting function; decimal block encoding; hexadecimal encoding; nibble encoding; modular compression; ternary constraint; residue classes modulo 10; residue classes modulo 3; lossless compression; Shannon entropy; popcount; wheel-based storage

---

1. Introduction

This note proposes a compact storage representation for primality based on decimal blocks. The starting point is elementary: every prime $p > 5$ satisfies

$$p \equiv 1, 3, 7, 9 \pmod{10}.$$

Therefore, for each decimal block

$$D_k = \{10k,\ 10k+1,\ \ldots,\ 10k+9\}, \qquad k \geq 1,$$

only the four candidates $10k+1$, $10k+3$, $10k+7$, $10k+9$ need to be stored.

The goal is not to introduce a new primality test or a sieve superior to classical methods. Rather, the purpose is to describe a compact block encoding of primality together with direct query algorithms. The resulting structure supports exact recovery, within the stored range, of the primality indicator, the prime-counting function $\pi(x)$, and the index of a stored prime via $\pi(p)$.

In Section 2 we define the encoding. Section 3 describes the database structure. Section 4 gives the core Python algorithms. Section 5 establishes the ternary constraint theorem and derives the compressed encoding. Section 6 reports experimental validation. Section 7 compares with traditional representations.

 doi:10.20944/preprints202604.0695.v1

2. Decimal–Hexadecimal Encoding

For each decade D_k, define four indicator bits

$$b_1(k),\ b_3(k),\ b_7(k),\ b_9(k) \in \{0,1\},$$

where

$$b_r(k) = \begin{cases} 1, & \text{if } 10k+r \text{ is prime,} \\ 0, & \text{otherwise,} \end{cases} \qquad r \in \{1,3,7,9\}.$$

We encode the block by the nibble

$$H(k) = 8\,b_1(k) + 4\,b_3(k) + 2\,b_7(k) + b_9(k),$$

so the bit order is $[1,3,7,9] \longleftrightarrow [8,4,2,1]$. Thus $H(k) \in \{0,1,\ldots,15\}$ and can be represented in hexadecimal.

Examples

$$10\text{–}19:\ 11,13,17,19 \text{ all prime } \Rightarrow\ 1111_2 = \texttt{F}.$$

$$20\text{–}29:\ 23,29 \text{ prime } \Rightarrow\ 0101_2 = \texttt{5}.$$

$$30\text{–}39:\ 31,37 \text{ prime } \Rightarrow\ 1010_2 = \texttt{A}.$$

$$40\text{–}49:\ 41,43,47 \text{ prime } \Rightarrow\ 1110_2 = \texttt{E}.$$

Hence the sequence begins `F,5,A,E,5,A,D,5,2,F,...`

3. Database Structure

The stored data consists of two components: (1) the base primes $2,3,5,7$, stored explicitly; (2) the sequence of nibbles $H(1), H(2), \ldots, H(K)$ for the desired range.

For compact storage, two consecutive nibbles are packed into one byte: high nibble = odd-indexed decade, low nibble = even-indexed decade. Hence two decades occupy one byte.

Proposition 1. *Given the explicitly stored primes* $2,3,5,7$ *and the sequence of nibbles* $H(k)$ *over a finite range, the primality indicator is exactly recoverable throughout that range.*

Proof. Every integer $n > 5$ is either outside the residue classes $1,3,7,9 \pmod{10}$, in which case it is automatically composite, or else belongs to a unique decade D_k and to exactly one of the four candidate positions. The corresponding bit of $H(k)$ records precisely whether that candidate is prime. Together with the explicit storage of $2,3,5,7$, this determines the primality indicator exactly in the stored range. □

4. Core Algorithms in Python

4.1. Construction of the Database

```
from math import isqrt

BASE_PRIMES = [2, 3, 5, 7]
ORDEN_BITS  = [1, 3, 7, 9]
PESOS       = [8, 4, 2, 1]

def es_primo(n):
    if n < 2: return False
    if n in (2, 3): return True
```

 doi:10.20944/preprints202604.0695.v1


```
    if n % 2 == 0: return False
    r = isqrt(n)
    d = 3
    while d <= r:
        if n % d == 0: return False
        d += 2
    return True

def nibble_decena(k, nmax):
    bits = []
    for r in ORDEN_BITS:
        n = 10 * k + r
        bits.append(1 if n <= nmax and es_primo(n) else 0)
    return sum(b * w for b, w in zip(bits, PESOS))

def construir_bd(nmax):
    kmax   = nmax // 10
    nibbles = [nibble_decena(k, nmax) for k in range(1, kmax + 1)]
    packed  = bytearray()
    i = 0
    while i < len(nibbles):
        hi = nibbles[i]
        lo = nibbles[i + 1] if i + 1 < len(nibbles) else 0
        packed.append((hi << 4) | lo)
        i += 2
    return nibbles, packed
```

4.2. Direct Primality Query

```
BIT_DE_RESIDUO = {1: 8, 3: 4, 7: 2, 9: 1}

def obtener_nibble(k, packed):
    byte_index = (k - 1) // 2
    byte = packed[byte_index]
    return (byte >> 4) & 0xF if k % 2 == 1 else byte & 0xF

def is_prime_from_db(n, packed):
    if n in BASE_PRIMES: return True
    if n < 2: return False
    r = n % 10
    if r not in BIT_DE_RESIDUO: return False
    k    = n // 10
    code = obtener_nibble(k, packed)
    return (code & BIT_DE_RESIDUO[r]) != 0
```

4.3. Recovery of the Prime-Counting Function

```
def popcount4(v):
    return ((v>>3)&1)+((v>>2)&1)+((v>>1)&1)+(v&1)

def prefix_popcounts(nibbles):
    pref = [0]; s = 0
    for v in nibbles:
        s += popcount4(v)
        pref.append(s)
```

 doi:10.20944/preprints202604.0695.v1


```
    return pref

def pi_from_db(x, packed, pref):
    if x < 2: return 0
    base = sum(1 for p in BASE_PRIMES if p <= x)
    if x <= 9: return base
    k     = x // 10
    total = 4 + pref[k - 1]
    nib   = obtener_nibble(k, packed)
    for r in ORDEN_BITS:
        if 10*k+r <= x and (nib & BIT_DE_RESIDUO[r]):
            total += 1
    return total
```

Proposition 2. *Within the stored range, $\pi(x)$ is exactly recoverable from the database by cumulative popcount together with the partial contribution of the block containing x.*

Proof. Each encoded decade contributes exactly $\mathrm{pc}(H(k))$ primes among the four admissible candidates. Summing these popcounts over all complete blocks before the block containing x, and adding the explicit contribution of $2, 3, 5, 7$ together with the partial count inside the current block, yields $\pi(x)$ exactly. □

5. The Ternary Constraint and Compressed Encoding

5.1. The Structural Theorem

The congruence $10 \equiv 1 \pmod 3$ implies

$$10k + r \equiv k + r \pmod 3.$$

Therefore $3 \mid (10k + r)$ if and only if $k \equiv -r \pmod 3$. Applying this to the four candidates:

Theorem 1 (Ternary Constraint)**.** *Let $k \geq 1$ and $r \in \{1, 3, 7, 9\}$. Then:*

1. *If $k \equiv 0 \pmod 3$, then $3 \mid (10k + 3)$ and $3 \mid (10k + 9)$, so $b_3(k) = b_9(k) = 0$.*
2. *If $k \equiv 1 \pmod 3$, no candidate is divisible by 3 (beyond the value 3 itself, which is handled in the base set).*
3. *If $k \equiv 2 \pmod 3$, then $3 \mid (10k + 1)$ and $3 \mid (10k + 7)$, so $b_1(k) = b_7(k) = 0$.*

Proof. Follows directly from $10k + r \equiv k + r \pmod 3$:

- $k \equiv 0$: $k + 3 \equiv 0$ and $k + 9 \equiv 0 \pmod 3$.
- $k \equiv 1$: $k + 1 \equiv 2$, $k + 3 \equiv 1$, $k + 7 \equiv 2$, $k + 9 \equiv 1 \pmod 3$; none zero.
- $k \equiv 2$: $k + 1 \equiv 0$ and $k + 7 \equiv 0 \pmod 3$.

Since all values $10k + r > 3$ for $k \geq 1$, divisibility by 3 implies compositeness. □

5.2. Partition of the Nibble Alphabet

Theorem 1 partitions the 16-symbol nibble alphabet into three disjoint effective alphabets:

Class	Forced zero bits	Effective alphabet
$k \equiv 0 \pmod 3$	b_3, b_9	$\{0, 2, 8, A\}$ (size 4)
$k \equiv 1 \pmod 3$	none	$\{0, \ldots, F\}$ (size 16)
$k \equiv 2 \pmod 3$	b_1, b_7	$\{0, 1, 4, 5\}$ (size 4)

 doi:10.20944/preprints202604.0695.v1

Corollary 1. *For $k \equiv 0$ or $k \equiv 2 \pmod 3$, the nibble $H(k)$ is determined by exactly 2 bits. For $k \equiv 1 \pmod 3$, all 4 bits are unconstrained.*

5.3. Shannon Entropy Reduction

Let p_r denote the empirical density of primes among the candidates of residue r in each class. By Dirichlet's theorem on primes in arithmetic progressions, all four residue classes $\{1, 3, 7, 9\}$ are asymptotically equiprobable, so the bits within the $k \equiv 1$ class are approximately i.i.d. with parameter $\hat{p} \approx 1/4$.

For the restricted classes ($k \equiv 0$ and $k \equiv 2$), only 2 bits are free, giving a maximum entropy of 2 bits; the observed Shannon entropy is approximately 1.82 bits due to the unequal probability of the symbol `0` (no primes in the active candidates).

The mean entropy per nibble is therefore

$$H = \tfrac{1}{3}H_0 + \tfrac{1}{3}H_1 + \tfrac{1}{3}H_2 \approx \tfrac{1}{3}(1.82) + \tfrac{1}{3}(3.64) + \tfrac{1}{3}(1.82) = 2.426 \text{ bits},$$

compared to the naive 4 bits, a reduction of **39.4**%.

Experimentally, over 30,000 decades up to 300,000, the measured entropy values are:

Class	H (bits)	Alphabet size
$k \equiv 0 \pmod 3$	1.8174	4
$k \equiv 1 \pmod 3$	3.6352	16
$k \equiv 2 \pmod 3$	1.8198	4
Weighted mean	2.4241	—

5.4. Compressed Encoding Scheme

Since the class of k modulo 3 is known implicitly from the position index, no overhead is required to signal which alphabet is in use. The practical encoding assigns:

- $k \equiv 0 \pmod 3$: a 2-bit index into $\{0, 2, 8, A\}$.
- $k \equiv 1 \pmod 3$: the original 4-bit nibble (or a Huffman code for further gain).
- $k \equiv 2 \pmod 3$: a 2-bit index into $\{0, 1, 4, 5\}$.

This yields an average of

$$\frac{1}{3}(2) + \frac{1}{3}(4) + \frac{1}{3}(2) = \frac{8}{3} \approx 2.667 \text{ bits per decade},$$

a lossless compression of 33.3% in a simple fixed-code scheme, approaching the Shannon limit of 39.4% if Huffman coding is applied to the $k \equiv 1$ class.

Remark 1. *The uniqueness of this compression is a direct consequence of $10 \equiv 1 \pmod 3$. The prime 3 is the only prime p for which $10 \equiv 1 \pmod p$, making it the unique prime that induces an exact periodic partition on the nibble sequence with period 3 and zero overhead. For the primes 5 and 7, the congruence $10 \not\equiv 1 \pmod p$ breaks this clean periodicity, and no analogous lossless partition exists at those moduli.*

```
# Compressed alphabets
ALPHA_K0 = [0x0, 0x2, 0x8, 0xA]    # k≡0 mod 3
ALPHA_K2 = [0x0, 0x1, 0x4, 0x5]    # k≡2 mod 3

IDX_K0 = {v: i for i, v in enumerate(ALPHA_K0)}
IDX_K2 = {v: i for i, v in enumerate(ALPHA_K2)}

def comprimir(nibbles):
```

 doi:10.20944/preprints202604.0695.v1


```
    """Encode nibble sequence using ternary constraint."""
    bits_out = []
    for k, v in enumerate(nibbles, start=1):
        mod = k % 3
        if mod == 0:
            idx = IDX_K0[v]
            bits_out.append(f"{idx:02b}")    # 2 bits
        elif mod == 1:
            bits_out.append(f"{v:04b}")      # 4 bits
        else:
            idx = IDX_K2[v]
            bits_out.append(f"{idx:02b}")    # 2 bits
    return "".join(bits_out)

def descomprimir(bitstring, n_decadas):
    """Decode compressed bitstring back to nibble sequence."""
    nibbles = []
    pos = 0
    for k in range(1, n_decadas + 1):
        mod = k % 3
        if mod == 1:
            v = int(bitstring[pos:pos+4], 2)
            pos += 4
        else:
            idx = int(bitstring[pos:pos+2], 2)
            v = ALPHA_K0[idx] if mod == 0 else ALPHA_K2[idx]
            pos += 2
        nibbles.append(v)
    return nibbles
```

6. Experimental Validation

6.1. Verification of the Ternary Constraint

Over all 30,000 decades up to 300,000, no nibble outside the prescribed alphabet was observed for $k \equiv 0$ or $k \equiv 2 \pmod 3$:

Nibble	$k \equiv 0$	$k \equiv 1$	$k \equiv 2$	Violation
0	4535	1914	4485	0
1	0	1036	2266	0
2	2235	1023	0	0
3	0	511	0	0
4	0	1061	2255	0
5	0	472	994	0
6	0	522	0	0
7	0	213	0	0
8	2214	1029	0	0
9	0	502	0	0
A	1016	506	0	0
B	0	218	0	0
C	0	513	0	0
D	0	206	0	0
E	0	205	0	0
F	0	69	0	0

 doi:10.20944/preprints202604.0695.v1

Zero violations confirm Theorem 1 exactly.

6.2. Size Comparison

Method	Size (bytes)	bits/decade
Explicit prime list (32-bit)	95,148	—
Full bitmap on $0, \ldots, 300{,}000$	37,500	—
Odd-only bitmap	18,750	—
Caraccioli original	15,000	4.000
Caraccioli + ternary (fixed 2/4/2 bits)	9,544	2.667
Caraccioli + ternary (Shannon limit)	9,090	2.424

7. Comparison with Traditional Representations

A full bitmap stores one bit per integer but wastes storage on even numbers and on decimal residue classes that cannot contain primes. An odd-only bitmap is already a reasonable baseline.

The original Caraccioli encoding reduces to 4 bits per decade by exploiting the residue structure mod 10. The ternary extension reduces this further to $8/3 \approx 2.67$ bits per decade (fixed scheme) or 2.42 bits per decade (entropy limit) by additionally exploiting the residue structure mod 3, with no loss of information and $O(1)$ encode/decode complexity.

The combined compression ratios relative to the original are:

- $1.65\times$ over the Caraccioli original (fixed scheme).
- $1.65\times$ to 39.4% improvement approaching the Shannon limit with Huffman coding on the $k \equiv 1$ class.
- $4.12\times$ over an odd-only bitmap.

8. Discussion

The ternary constraint is not a heuristic observation: it is a theorem that follows from $10 \equiv 1 \pmod 3$ alone, and it holds for every $k \geq 1$ without exception. Its consequence — that two thirds of all nibble positions are confined to a 4-symbol alphabet — is a direct structural fact about the distribution of primes modulo 30, since the admissible residues mod 30 are exactly those coprime to 30.

The compression gain is genuine and practically implementable. Decoding requires only knowing k mod 3, which is free from the position index. No auxiliary table beyond the two 4-element alphabets is required.

A natural further extension is the wheel modulo $210 = 2 \cdot 3 \cdot 5 \cdot 7$, which would yield 8 admissible residues per block. However, since $10 \not\equiv 1 \pmod 5$ and $10 \not\equiv 1 \pmod 7$, the corresponding constraints do not produce clean periodic partitions of the nibble sequence and require a more involved analysis.

9. Conclusions

We presented a decimal–hexadecimal block encoding for primality (Caraccioli encoding) and established a structural theorem — the Ternary Constraint — that partitions the 16-symbol nibble alphabet into three effective alphabets of sizes 4, 16, and 4 according to k mod 3.

This partition yields:

- a provable lossless compression of 33.3% (fixed scheme) to 39.4% (Shannon limit) over the original encoding,
- $O(1)$ encode and decode complexity,
- a combinatorial explanation for the dominant period-3 autocorrelation peak observed in the nibble sequence,

 doi:10.20944/preprints202604.0695.v1

- the observation that $p = 3$ is the unique prime inducing this exact periodic structure, by virtue of $10 \equiv 1 \pmod 3$.

1. G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, 6th ed., Oxford University Press, 2008.
2. R. Crandall and C. Pomerance, *Prime Numbers: A Computational Perspective*, 2nd ed., Springer, 2005.
3. D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, 1997.
4. T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed., Wiley, 2006.