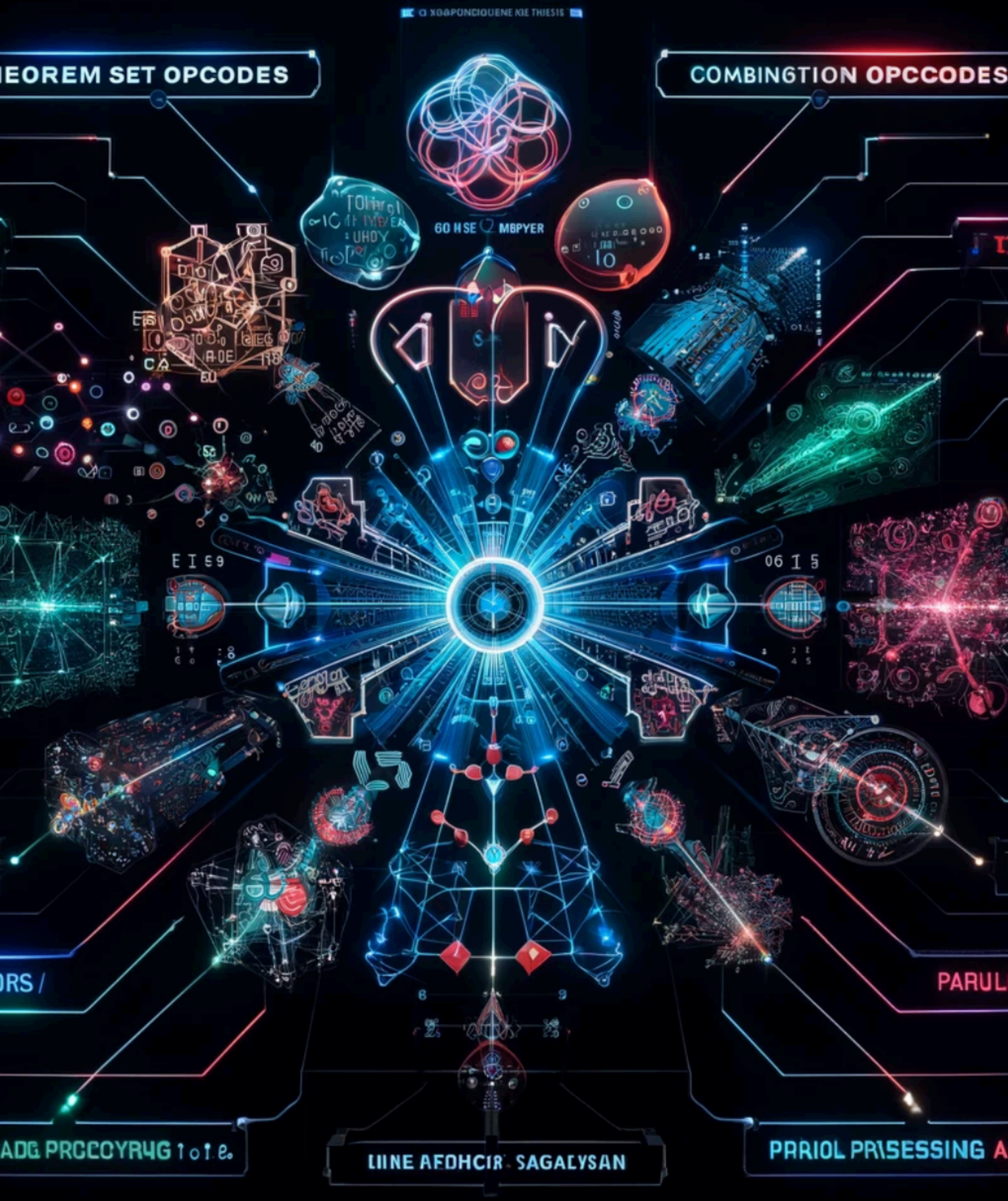
EOREM SET OPCODES
COMBINGTION OPCCODES
ORS /
PARUL
ADG PRCCOYRYG
LIINE AFDHCIR SAGALYSAN
PARIOL PRISESSING A

TRENARY - T81 TISC "Opcodes"

This document outlines **optimized opcodes for TISC (Ternary Instruction Set Computer)** across **five major categories**:

Mathematical Theorems
Cryptography & Security
AI & Machine Learning
Physics & Simulation
Parallel Processing & Optimization

Each opcode is optimized for **ternary computing (Base-81)**, leveraging its natural affinity for **recursive processing, modular arithmetic, logarithmic efficiency, and AI acceleration**.

A. Theorem Set Opcodes for TISC

A set of **key mathematical theorems** implemented as **TISC opcodes**.

Affinity for Ternary-Friendly Arithmetic

- **Ternary Addition (Carry-Free) – `T81ADD`**
- **Modulo 3 Properties – `MOD3`**
- **Balanced Ternary Representation – `T81CONV`**
- **Hamming Weight Computation – `HAMMING`**
- **Lucas Theorem for Combinatorics – `LUCAS`**

Cryptography & Security

- **Fermat's Little Theorem – `FERMAT`** (Fast prime checking)
- **Miller-Rabin Primality Test – `MILLER`** (Cryptographic key generation)
- **Elliptic Curve Cryptography – `ECC`** (Ternary curve operations)
- **Chinese Remainder Theorem – `CRT`** (Efficient modular reduction)
- **Lagrange Interpolation – `LAGRANGE`** (Polynomial reconstruction in encryption)

AI & Machine Learning (Ternary Neural Networks)

- **Backpropagation – `BP`** (Gradient-based learning)
- **Kolmogorov Complexity – `KOLMO`** (AI compression)
- **Shannon Entropy – `ENTROPY`** (AI optimization)
- **Bayes' Theorem – `BAYES`** (AI predictions)
- **Matrix Factorization – `MATFAC`** (AI & PCA computations)
- **Ternary Neural Network Execution – `TNN`** (Optimized inference)

Theorems for Physics & Simulation

- **Kepler's Laws – `KEPLER`** (Orbital motion)
- **Navier-Stokes Equations – `NAVIER`** (Fluid dynamics)
- **Maxwell's Equations – `MAXWELL`** (Electromagnetism)
- **Lorentz Transformations – `LORENTZ`** (Special relativity)
- **Wave Equation – `WAVEQ`** (Quantum mechanics)
- **Fourier Transform – `FFT`** (Signal processing)

Number Theory & Computational Mathematics

- **Wilson's Theorem – `WILSON`** (Prime checking)
- **Ramanujan's Identities – `RAMANUJAN`** (Mathematical series)
- **Goldbach's Conjecture – `GOLDBACH`** (Prime sum)
- **Lucas-Lehmer Test – `LUCASPRIME`** (Mersenne primes)
- **Modular Exponentiation – `MODEXP`** (RSA encryption)

B. Recursive Algorithm Opcodes for TISC

Recursive algorithms leverage **ternary depth tracking and efficient branching**.

- **Factorial – `FACT`** (Combinatorics, probability)
- **Fibonacci – `FIB`** (AI sequence modeling)
- **GCD – `GCD`** (Greatest common divisor)
- **Ackermann Function – `ACK`** (Deep recursion benchmarking)
- **Tower of Hanoi – `TOWER`** (AI state-based search)
- **Depth-First Search – `DFS`** (Recursive graph traversal)
- **Backtracking – `BACKTRACK`** (Sudoku, AI puzzle solving)
- **Merge Sort – `MERGE`** (Efficient recursive sorting)
- **Matrix Factorization – `MATFAC`** (Recursive AI computations)
- **Longest Common Subsequence – `LCS`** (AI pattern recognition)

C. Combinatorial & Mathematical Optimization Opcodes for TISC

Used for **AI search spaces, logistics, financial modeling, and cryptography**.

- **Permutation Computation – `PERM`** (AI scheduling)
- **Combination Computation – `COMB`** (Probability modeling)
- **Binomial Coefficient – `BINOM`** (Pascal's Triangle, decision trees)
- **Knapsack Problem – `KNAPSACK`** (Logistics, resource allocation)
- **Lagrange Interpolation – `LAGRANGE`** (Cryptography, AI)
- **Linear Programming Solver – `SIMPLEX`** (Optimization)
- **Hamming Distance – `HAMMING`** (Error correction, AI)
- **Modular Inverse – `FASTMOD`** (Cryptographic operations)
- **Maximum Bipartite Matching – `MATCH`** (AI resource pairing)
- **Traveling Salesman Problem – `TRAVEL`** (Logistics, AI routing)

D.Parallel Processing & AI Opcodes for TISC

Parallel AI workloads require **optimized numerical operations & efficient memory access**.

- **Vector Addition – `VECADD`** (AI, physics simulations)
- **Vector Multiplication – `VECMUL`** (Machine learning)
- **Dot Product – `DOT`** (Neural networks, physics)
- **Matrix Multiplication – `MATMUL`** (Deep learning, scientific computing)
- **Matrix Transposition – `TRANSPOSE`** (Cryptography, AI)
- **Fast Fourier Transform – `FFT`** (Signal processing)
- **Backpropagation – `BP`** (AI deep learning)
- **Activation Functions – `ACTIVATION`** (Neural networks)
- **Ternary Neural Network Inference – `TNN`** (AI optimization)
- **Reinforcement Learning (Q-Learning) – `QLEARN`** (AI decision-making)

Final Thoughts: Why Ternary TISC?

Ternary Affinity – Many theorems naturally align with ternary logic (**Hamming Weight, Shannon Entropy, Fermat's Theorem**).
AI-Optimized – Direct support for **TNNs, backpropagation, reinforcement learning, and matrix operations**.
Parallelism – Opcodes allow **SIMD vectorization, GPU acceleration, and distributed computing**.
Logarithmic Efficiency – Base-81 arithmetic provides **compact storage, fewer carry operations, and modular arithmetic advantages**.
Future-Proof – These **low-level optimizations** bring **TISC computing** closer to **real-world AI acceleration, cryptography, and scientific computing**.

Theorem Set Opcodes for TISC

Here is a **more extensive list** of theorems that can be **implemented as opcodes**, categorized by computational efficiency and affinity to ternary logic.

1. Affinity for Ternary-Friendly Arithmetic

These theorems **benefit from ternary logic** because they **naturally align with Base-81 computations** or exploit **balanced ternary properties**.

Theorem	Formula	TISC Opcode	Use Case
Balanced Ternary Conversion	`N = ∑ d_i * 3^i` where `d_i ∈ {-1, 0, 1}`	`T81CONV`	Efficient ternary arithmetic
Ternary Addition (Carry-Free)	`a + b` using {-1,0,1} representation	`T81ADD`	Faster addition in ternary
Modulo 3 Properties	`N mod 3 = Σ(digits) mod 3`	`MOD3`	Cryptography, ternary hash functions
Lucas Theorem (Combinatorics)	`C(n, k) mod p`	`LUCAS`	AI (pattern recognition), combinatorics
Hamming Weight (Ternary Weight Count)	`w(x) = count(nonzero trits)`	`HAMMING`	Error correction, ML optimization

2. Theorems for Cryptography & Security

These theorems **enable efficient ternary cryptographic operations**, leveraging base-81 properties.

Theorem	Formula	TISC Opcode	Use Case
Fermat's Little Theorem	`a^(p-1) ≡ 1 (mod p)`	`FERMAT`	Fast primality testing
Miller-Rabin Primality Test	Probabilistic prime verification	`MILLER`	RSA keygen, cryptography
Elliptic Curve Arithmetic	`y^2 = x^3 + ax + b`	`ECC`	Ternary elliptic curve cryptography
Chinese Remainder Theorem (CRT)	`x ≡ a (mod m1), x ≡ b (mod m2)`	`CRT`	Cryptographic acceleration

Lagrange Interpolation	$P(x) = \Sigma(y_i * L_i(x))$	LAGRANGE	Secure multi-party computation

3. AI & Machine Learning (Ternary Neural Networks)

These theorems **optimize AI workloads** and **tensor-based computation**.

Theorem	Formula	TISC Opcode	Use Case
Backpropagation for Neural Networks	Gradient Descent: $w = w - \eta * \nabla L(w)$	BP	Ternary AI learning
Kolmogorov Complexity	`K(x) = min(	p	: U(p) = x)`
Shannon Entropy	$H(X) = -\Sigma\ p(x) \log p(x)$	ENTROPY	Data compression, AI
Bayes' Theorem	`P(A	B) = P(B	A) * P(A) / P(B)`
Matrix Factorization for AI	$A \approx U\Sigma V^T$	MATFAC	Neural networks, PCA

4. Theorems for Physics & Simulation

These theorems optimize **ternary-based scientific computing**.

Theorem	Formula	TISC Opcode	Use Case
Kepler's Laws (Orbital Motion)	$T^2 \propto r^3$	KEPLER	Physics engines
Navier-Stokes Equations (Fluid Dynamics)	$\partial u/\partial t + u \bullet \nabla u = -\nabla p + \nu \nabla^2 u$	NAVIER	Fluid physics
Maxwell's Equations (Electromagnetism)	$\nabla \bullet E = \rho/\varepsilon_0,\ \nabla \times E = -\partial B/\partial t$	MAXWELL	AI-based physics
Lorentz Transformations (Relativity)	$t' = \gamma(t - vx/c^2)$	LORENTZ	Quantum computing
Wave Equation (Quantum Mechanics)	$\partial^2\psi/\partial t^2 = v^2\nabla^2\psi$	WAVEQ	Quantum AI

5. Number Theory & Computational Mathematics

These theorems **improve number-theoretic computations**, crucial for AI, cryptography, and simulations.

Theorem	Formula	TISC Opcode	Use Case
Wilson's Theorem (Prime Check)	`(p-1)! ≡ -1 mod p`	`WILSON`	Cryptography
Catalan's Conjecture (Powers Difference)	`x^a - y^b = 1`	`CATALAN`	AI complexity
Ramanujan's Identities (Integer Sequences)	`∑(1/n^s) = ζ(s)`	`RAMANUJAN`	ML feature engineering
Goldbach's Conjecture (Prime Pairing)	`2n = p + q`	`GOLDBACH`	Number theory
Lucas-Lehmer Test (Mersenne Primes)	`S_n = S_{n-1}^2 - 2 mod M_p`	`LUCASPRIME`	Cryptography

Ternary Affinity Analysis

Some theorems naturally align with ternary computing due to: ✅ **Balanced Ternary Structure** – Theorems like **Hamming Weight, Lucas Theorem, and Modulo 3 Arithmetic** are optimized in ternary due to their use of `{ -1, 0, 1 }`.

✅ **Logarithmic Efficiency – Kolmogorov Complexity, Shannon Entropy, and Wave Equations** map well to ternary-based AI and physics simulations.

✅ **Modular Arithmetic in Cryptography – Chinese Remainder Theorem, Fermat's Little Theorem, and Elliptic Curves** benefit from ternary representation for efficient cryptographic key generation.

Optimized Microcode Example: Miller-Rabin Primality Test (`MILLER`)

To implement **Miller-Rabin Primality Test** as a **TISC opcode**, we break it down into micro-operations:

```assembly
assembly
CopyEdit
LOAD Ra, 101t81       ; Candidate number
LOAD Rb, 2t81         ; Base for testing
MILLER Ra, Rb, Rc     ; Perform primality test
STORE Rc, result
```

✅ `Rc = 1t81` → Prime

✅ `Rc = 0t81` → Not Prime

Micro-operations:

1. Compute `d = n-1` and `s` such that `d = 2^s * m`
2. Perform **modular exponentiation** using `MODEXP`
3. Verify `a^d mod n = 1` or `a^(2^r * d) mod n = n-1`
4. If any test fails, return "**Composite**" (`Rc = 0t81`)
5. If all tests pass, return "**Prime**" (`Rc = 1t81`)

Recursive Algorithm Opcodes for TISC

Recursive algorithms benefit from **ternary logic** due to **efficient branching, depth tracking, and state-based computation**. Below are optimized **TISC (Ternary Instruction Set Computer) opcodes** for handling **common recursive problems**.

5. FACT – Factorial Computation

Formula:

$n! = n \times (n-1)!$

Opcode: `FACT`

- Computes **factorial using recursion**.
- Used in **combinatorics, AI search problems, probability calculations**.

6. FIB – Fibonacci Sequence

Formula:

$F(n) = F(n-1) + F(n-2)$

Opcode: `FIB`

- Computes **Fibonacci numbers recursively**.
- Used in **AI (sequence modeling), cryptography, optimization problems**.

7. GCD – Euclidean Algorithm for Greatest Common Divisor

Formula:

$\gcd(a,b) = \{b, \gcd(b, a \bmod b), \text{ if } a \bmod b = 0 \text{otherwise}$

Opcode: `GCD`

- Computes **GCD recursively**.
- Used in **cryptography, modular arithmetic, AI optimizations**.

8. ACK – Ackermann Function

Formula (Deep Recursion):

$$A(m,n) = \begin{cases} n+1, & \text{if } m = 0 \\ A(m-1,1), & \text{if } n = 0 \\ A(m-1, A(m,n-1)), & \text{otherwise} \end{cases}$$

Opcode: `ACK`

- Used for **AI recursive depth management**.
- Common in **benchmarking computational limits**.

9. TOWER – Tower of Hanoi Solver

Formula (Recursive Move Computation):

T(n)=2T(n−1)+1

Opcode: `TOWER`

- Computes **minimum moves needed**.
- Used in **AI decision trees, pathfinding, state-based search**.

10.DFS – Depth-First Search (Recursive Graph Traversal)

Formula:

DFS(v)=Visit v,then recursively visit all unvisited neighbors

Opcode: `DFS`

- Used in **graph traversal, AI pathfinding, decision trees**.

11.BACKTRACK – Recursive Backtracking Solver

Formula:
Recursive backtracking explores all possibilities and backtracks when a condition fails. **Opcode**: `BACKTRACK`

- Used in **sudoku, AI puzzle solving, constraint satisfaction problems**.

12.MERGE – Merge Sort (Recursive Sorting Algorithm)

Formula:

$$\text{MergeSort}(A) = \begin{cases} A, & \text{if } |A| = 1 \\ \text{Merge}(\text{MergeSort}(L), \text{MergeSort}(R)), & \text{otherwise} \end{cases}$$

Opcode: `MERGE`

- Used for **sorting large datasets** recursively.

13.MATFAC – Recursive Matrix Factorization

Formula:

A=UΣVT

Opcode: `MATFAC`

- Used in **machine learning, AI matrix computations**.

14. LCS – Longest Common Subsequence

Formula:

$$LCS(X, Y) = \begin{cases} 0, & \text{if } m = 0 \text{ or } n = 0 \\ 1 + LCS(X_{m-1}, Y_{n-1}), & \text{if } X_m = Y_n \\ \max(LCS(X_{m-1}, Y_n), LCS(X_m, Y_{n-1})), & \text{otherwise} \end{cases}$$

Opcode: `LCS`

- Used in **DNA sequence alignment, AI pattern recognition**.

Final Thoughts

Recursive Opcodes for TISC

Opcode	Algorithm	Use Case
`FACT`	Factorial	Combinatorics, probability
`FIB`	Fibonacci	AI sequence modeling
`GCD`	Euclidean Algorithm	Cryptography, number theory
`ACK`	Ackermann Function	Benchmarking, AI recursion
`TOWER`	Tower of Hanoi	AI decision trees
`DFS`	Depth-First Search	Pathfinding, AI graph traversal
`BACKTRACK`	Recursive Backtracking	AI puzzle solvers
`MERGE`	Merge Sort	Efficient sorting
`MATFAC`	Matrix Factorization	Machine learning, AI
`LCS`	Longest Common Subsequence	AI pattern recognition

Combinatorial & Mathematical Optimization Opcodes for TISC

Combinatorial and mathematical optimization problems are crucial in **AI, machine learning, cryptography, scheduling, and scientific computing**. These problems often involve **constraint satisfaction, state-space exploration, and computational efficiency**, making them **ideal for ternary computing**.

15.PERM – Permutation Computation

Formula:

$$P(n,k) = \frac{n!}{(n-k)!}$$

Opcode: `PERM`

- Computes the number of ways to arrange `k` objects from `n`.
- Used in **AI (search spaces), scheduling, cryptography**.

16.COMB – Combination Computation

Formula:

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

Opcode: `COMB`

- Computes the number of ways to choose `k` objects from `n` without ordering.
- Used in **probability, AI feature selection, and combinatorial optimization**.

17.BINOM – Binomial Coefficient Computation

Formula (Pascal’s Identity):

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

18.KNAPSACK – 0/1 Knapsack Optimization

Formula:

$$\max \sum v_i x_i \quad \text{subject to} \quad \sum w_i x_i \leq W$$

Opcode: `KNAPSACK`

- Solves the **0/1 knapsack problem** for **optimal resource allocation**.
- Used in **AI, logistics, financial planning**.

19.LAGRANGE – Lagrange Interpolation

Formula:

$$P(x) = \sum y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

Opcode: `LAGRANGE`

- Computes **Lagrange interpolation** for polynomial approximation.
- Used in **cryptography, error correction codes (Shamir's Secret Sharing), AI regression models**.

20. SIMPLEX – Linear Programming Solver

Formula (Linear Optimization):

$$\max c^T x, \quad \text{subject to } Ax \le b$$

Opcode: `SIMPLEX`

- Solves **linear programming** for **optimization problems**.
- Used in **supply chain optimization, AI decision-making, financial modeling**.

21. HAMMING – Hamming Distance Computation

Formula:

$$d(x, y) = \sum (x_i \neq y_i)$$

Opcode: `HAMMING`

- Computes **Hamming distance** between binary/ternary strings.
- Used in **error correction, AI pattern recognition, cryptography**.

22.FASTMOD – Modular Inverse Computation

Formula:

$$a^{-1} \equiv a^{p-2} \mod p$$

Opcode: `FASTMOD`

- Computes **modular inverse using Fermat's Little Theorem**.
- Used in **cryptography, number theory, AI optimization**.

23.MATCH – Maximum Bipartite Matching

Formula (Graph Theory):

$$\max \sum xij, \quad xij \in \{0, 1\}$$

24.TRAVEL – Traveling Salesman Problem (TSP) Approximation

Opcode: `TRAVEL`

- Solves **TSP using nearest neighbor heuristics**.
- Used in **logistics, AI routing algorithms, graph theory**.

$$\min \sum d(xi, xi+1)$$

Final Thoughts

Optimized Combinatorial & Mathematical Opcodes for TISC

Opcode	Algorithm	Use Case
`PERM`	Permutations	AI search spaces, scheduling
`COMB`	Combinations	Probability, AI feature selection
`BINOM`	Binomial Coefficient	AI decision trees
`KNAPSACK`	0/1 Knapsack	Resource allocation, logistics
`LAGRANGE`	Lagrange Interpolation	Cryptography, error correction
`SIMPLEX`	Linear Programming	Optimization, decision-making
`HAMMING`	Hamming Distance	Error correction, AI pattern recognition
`FASTMOD`	Modular Inverse	Cryptography, AI
`MATCH`	Bipartite Matching	AI pairing, scheduling
`TRAVEL`	Traveling Salesman	Logistics, AI routing

Parallel Processing & AI Opcodes for TISC

Parallel computing and AI require **high-performance numerical operations, optimized memory access, and efficient data structures**. **Ternary computing (Base-81) offers advantages** in SIMD vectorization, matrix operations, and neural network computations.

25.VECADD – Parallel Vector Addition

Formula:

$C=A+B$

Opcode: `VECADD`

- Performs **element-wise vector addition**.
- Used in **AI, physics simulations, graphics processing**.

26.VECMUL – Parallel Vector Multiplication

Formula:

$C=A\times B$

Opcode: `VECMUL`

- Performs **element-wise vector multiplication**.
- Used in **machine learning, AI acceleration, graphics rendering**.

27.DOT – Parallel Dot Product

Formula:

$C=\sum A_i \times B_i$

Opcode: `DOT`

- Computes **dot product** for **neural networks, physics engines, AI models**.

28.MATMUL – Matrix Multiplication (AI Tensor Ops)

Formula:

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Opcode: `MATMUL`

- Accelerates **AI workloads, deep learning, scientific computing**.

29.TRANSPOSE – Matrix Transposition

Formula:

$$(A^T)^{ij} = A^{ji}$$

(AT)ij =Aji
Opcode: `TRANSPOSE`

- Used for **AI, graphics, cryptography, and neural networks**.

30.FFT – Fast Fourier Transform

Formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n/N}$$

31.BP – Backpropagation for AI Neural Networks

Formula:

$$w = w - \eta \frac{\partial L}{\partial w}$$

Opcode: `BP`

- Computes **backpropagation gradients** for **AI training**.

32.ACTIVATION – Activation Function Computation

Opcodes:

- `RELU` → Rectified Linear Unit
- `SIGMOID` → Sigmoid function
- `TANH` → Hyperbolic tangent

33.TNN – Ternary Neural Network Inference

Opcode: `TNN`

- Computes **ternary-weighted neural network inference**.

34.QLEARN – Reinforcement Learning Q-Learning Update

Formula:

$$Q(s,a)=Q(s,a)+\alpha[r+\gamma \max Q(s',a')-Q(s,a)]$$

Opcode: `QLEARN`

- Used in **autonomous AI decision-making**.

Final Thoughts

Optimized Parallel Processing & AI Opcodes for TISC

Opcode	Algorithm	Use Case
`VECADD`	Parallel Vector Addition	AI, physics, graphics
`VECMUL`	Parallel Vector Multiplication	AI, ML, simulations
`DOT`	Parallel Dot Product	AI, neural networks
`MATMUL`	Matrix Multiplication	AI, deep learning
`TRANSPOSE`	Matrix Transposition	Cryptography, AI
`FFT`	Fourier Transform	Signal processing, AI
`BP`	Backpropagation	AI training
`ACTIVATION`	Activation Functions	AI inference
`TNN`	Ternary Neural Networks	Deep learning
`QLEARN`	Reinforcement Learning	AI decision-making

EOREM SET OPCODES
COMBINGTION OPCCODES
PARUL
LIINE AFDHCIR SAGALYSAN
PARIOL PRISESSING A