DIRAM Boolean Logic Truth Table - Memory Management Gates

OBINexus Aegis Project | Directed Instruction RAM

Governance Constraint: $\varepsilon(x) \leq 0.5$ | **Binary Logic**: 2-Input, 1-Output

---

⚙️ Logic Gate Truth Table Breakdown

Here we're looking at how binary inputs transform into actionable output using gates like NOT and XOR:

Input A	Input B	NOT A	A XOR B	Final Output
0	0	1	0	**1**
0	1	1	1	**0**
1	0	0	1	**1**
1	1	0	0	**0**

🧠 Memory Management + Binary State

Input Definitions:

- **Input A**: Cache State (0 = Cache Miss, 1 = Cache Hit)
- **Input B**: Governance State (0 = ε≤0.5 Compliant, 1 = ε>0.5 Violation)
- **Final Output**: Memory Action (0 = Block/Defer, 1 = Allow/Process)

Truth Table Logic Explanation:

Row 1: A=0, B=0 → Cache Miss + Compliant

- NOT A = 1 (miss requires action)
- XOR = 0 (both inputs low)
- **Output = 1** ✅ **Allow**: Cache miss with good governance → Fetch data, update cache

Row 2: A=0, B=1 → Cache Miss + Violation

- NOT A = 1 (miss requires action)
- XOR = 1 (inputs differ)
- **Output = 0** ❌ **Block**: Cache miss during constraint violation → Defer allocation

Row 3: A=1, B=0 → Cache Hit + Compliant

- NOT A = 0 (hit needs no extra action)
- XOR = 1 (inputs differ)

- **Output = 1** ✅ **Allow**: Cache hit with good governance → Process immediately

Row 4: A=1, B=1 → Cache Hit + Violation

- NOT A = 0 (hit needs no extra action)
- XOR = 0 (both inputs high)
- **Output = 0** ❌ **Block**: Even cache hits blocked during severe violations

---

🔁 Cache Hits vs Misses (Lookahead Memory Logic)

When your system needs data, it looks in cache first (like a quick-access drawer). Two things can happen:

Cache Hit 🟢

The needed data is already there—no extra fetch needed. System stays fast.

- **Example**: Permitted data is preloaded and the signal finds it instantly
- **Triggers Update**: Memory confirms access, adjusts state, nudges related predictions
- **LRU/MRU Action**: Promotes accessed item to Most Recently Used

Cache Miss 🔴

The drawer's empty! Now the system must dig deeper (main memory or disk).

- **Data wasn't updated** into cache beforehand, so no immediate response
- **Lookup fails**, slowing things down until fresh info loads
- **LRU Action**: Must evict Least Recently Used item to make space

Lookahead Hardware Prediction

Tries to predict future cache needs—preloading data it suspects the system will ask for. If prediction aligns, more hits happen.

🎯 Governance Constraint: ε(x) ≤ 0.5

Sinphasé Governance Model:

```c
bool diram_check_sinphase_compliance(uint8_t heap_events, uint8_t max_events) {
    double epsilon = (double)heap_events / (double)max_events;
    return epsilon <= 0.5;  // Updated constraint (not 0.6)
}
```

Governance States:

- **B = 0**: ε ≤ 0.5 → System running within safe memory allocation limits
- **B = 1**: ε > 0.5 → Too many heap events, system must throttle allocations

---

📊 Memory Hardware Address Layout

The gates act like **checkpoints**: deciding when binary info should be stored, passed through, or flipped.

LRU (Least Recently Used) Logic:

```
Cache Full? → Need Eviction → Check LRU Chain → Remove Oldest
```

MRU (Most Recently Used) Logic:

```
Cache Hit? → Promote Item → Move to MRU Position → Update Chain
```

DIRAM Traceable Cache:

- **Cache hit** often aligns with predictable output patterns (like repeated 1s)
- **Cache miss** comes from unpredictable or rare signal paths—where XOR flips unexpectedly or NOT cancels out expected inputs
- **SHA-256 receipts** generated for every cache operation
- **Lookahead prediction** uses confidence scoring to preload likely data

---

🔧 Hardware Implementation

```c
#include "diram"

// Binary decision function
uint8_t diram_memory_gate(uint8_t cache_state, uint8_t governance_state) {
    uint8_t not_a = !cache_state;
    uint8_t xor_ab = cache_state ^ governance_state;

    // Truth table logic: various combinations based on requirements
    return (not_a && !xor_ab) || (!not_a && xor_ab);
}
```

Cache Layout for New Algorithms:

- **Address tracing**: Hardware can see cache layout patterns
- **LRU/MRU transitions**: Binary decisions based on access patterns

- **Predictive allocation**: Uses historical patterns to forecast future needs
- **Governance enforcement**: ε(x) ≤ 0.5 constraint checked at hardware level

---

🏗️ Memory Evolution: Random → Directed

Traditional RAM: Passive storage responding to requests **DIRAM**: Active memory making intelligent decisions based on:

- Binary logic gates for fast decision-making
- Cache hit/miss prediction patterns
- Governance constraints preventing resource exhaustion
- Cryptographic traceability for security

The truth table shows how **2 simple binary inputs** can create sophisticated memory management behavior through careful logic gate design.

Result: Memory that doesn't just store—it **thinks, predicts, and governs** its own allocation patterns using boolean logic as the foundation.