Execution Comparison Results: C vs Zig Emulators

Date: 2026-01-12 20:00, Updated 2026-01-15 13:17 **Status**: In Progress - Zig UNBIND crash fixed; Zig now reaches instruction 6 (22 lines) **Purpose**: Document execution comparison results between C and Zig emulators

Overview

After fixing the C emulator PC advancement bug and implementing GVAR BIGATOMS mode in Zig, both emulators now execute identically for the first 5 instructions. This document tracks the comparison results and identifies remaining issues.

Comparison Methodology

Log Generation

Both emulators generate execution logs with unified format:

- C: `c_emulator_execution_log.txt` (1000 instructions)
- Zig: `zig_emulator_execution_log.txt` (stops on crash/error)

Comparison Process

1. Generate logs from both emulators
2. Compare line by line for PC, instruction, stack, and frame values
3. Identify first divergence point
4. Analyze bit-shifted values to find root cause

Results Summary

First 5 Instructions: ✅ Perfect Match (All Traced and Verified)

Line	C PC	C Instruction	Zig PC	Zig Instruction	Status
1	0x60f130	POP	0x60f130	POP	✅ Traced
2	0x60f131	GVAR	0x60f131	GVAR	✅ Traced, Fixed (BIGATOMS)
3	0x60f136	UNBIND	0x60f136	UNBIND	✅ Traced, Fixed (offset)
4	0x60f137	GETBASEPTR_N	0x60f137	GETBASEPTR_N	✅ Traced, Fixed (byte order)
5	0x60f139	COPY	0x60f139	COPY	✅ Traced, Verified
6	0x60f13a	TJUMP1	0x60f13a	TJUMP1	✅ Traced, Fixed (offset 3)

Tracing Documents:

- `c-emulator-address-xor-tracing.typ` - GVAR XOR addressing
- `c-emulator-unbind-tracing.typ` - UNBIND stack unwinding
- `c-emulator-getbaseptr-tracing.typ` - GETBASEPTR_N memory access
- `c-emulator-copy-tracing.typ` - COPY stack duplication

GVAR PC Advancement: ✅ Fixed

- C emulator: Advances PC by 5 bytes (`0x60f131 → 0x60f136`)
- Zig emulator: Now advances PC by 5 bytes (matches C)
- Root cause: Zig was using 2-byte atom numbers instead of 4-byte pointers
- Fix: Updated to BIGATOMS+BIGVM mode (5-byte instruction length)

Remaining Issues

Zig Emulator Crash at Instruction 3 (UNBIND) - 2026-01-14 (Resolved 2026-01-15)

Status: ✅ Fixed - No longer blocks log comparison

After refactoring the C emulator tracing code into `maiko/src/tracing/`, we tested log generation:

C Emulator:

- ✅ Successfully generates 1000-line execution log
- ✅ Log file: `c_emulator_execution_log.txt`
- ✅ Starting PC: `0x60f130` (matches expected)
- ✅ Complete log format with all fields

Zig Emulator:

- ✅ UNBIND no longer crashes at instruction 3
- ✅ Generates 22 lines (reaches at least instruction 6: TJUMP1)
- ✅ Starting PC: `0x60f130` (matches C emulator)
- ⚠️ Log format still needs tightening to be readily comparable to C (some fields appear truncated)

Crash Details:

- Location: `zaiko/src/vm/dispatch/execution_control.zig:110`
- Function: `opcodes.handleUNBIND(vm)`
- Stack trace: `handleControlFlow` → `routeOpcode` → `executeOpcodeWithOperands` → `executeInstruction` → `executeInstructionInLoop` → `dispatch`

Comparison Findings:

- Instruction 1 (POP): ✅ PC matches (`0x60f130`), but Zig log format incomplete
- Instruction 2 (GVAR): ✅ PC matches (`0x60f131`), but stack values differ (TOS differs)
- Instruction 3 (UNBIND): ✅ PC matches (`0x60f136`), but Zig crashes before completing

Fix Summary (Zig):

- Root cause: Zig UNBIND marker search did not match C's `*--CSTKPTRL` semantics (decrement-first, then read) and hit alignment traps when casting `(DLword *)PVAR + ...` to `LispPTR *`.
- Fix: In `zaiko/src/vm/opcodes/binding.zig`, UNBIND now decrements the stack pointer first and reads the marker value directly from stack memory (byte-swapped), and uses `[*]align(1) LispPTR` for potentially-unaligned `ppvar`.

Next Steps:

1. Fix Zig log format to match C format (complete fields; no truncation)
2. Re-run C vs Zig comparison from instruction 1 onward (now that UNBIND executes)
3. Investigate why TOS differs at instruction 2 (GVAR)

Zig Emulator Crash After 48 Instructions (Previous Issue)

Status: ⚠️ Superseded by instruction 3 crash

The Zig emulator previously crashed after approximately 48 instructions with `error.InvalidAddress`. This issue is now superseded by the earlier crash at instruction 3

Possible Causes:

1. Invalid address calculation in atom lookup
2. Memory access beyond virtual memory bounds
3. Incorrect address translation for atom pointers
4. Stack corruption causing invalid pointer dereference

Next Steps:

1. Add detailed tracing around instruction 48
2. Compare stack and frame state at crash point
3. Verify atom pointer values and address translation

4. Check for memory bounds violations

Key Findings

C Emulator PC Advancement Bug

Issue: PC was not being updated from `pccache` after opcode execution **Fix**: Added `PC = PCMAC` update before resetting `pccache` **Impact**: Enabled proper execution and comparison

GVAR BIGATOMS Mode

Issue: Zig assumed 2-byte atom numbers, C uses 4-byte pointers **Fix**: Updated instruction length to 5 bytes and added `getPointerOperand()` **Impact**: First 5 instructions now match exactly

Unified Logging Format

Implementation: Both emulators use identical log format with hex+octal+bit-shifts **Benefit**: Enables precise comparison and off-by-one-bit error detection

Verification Status

✅ C emulator executes correctly (source of truth) ✅ First 5 instructions match between C and Zig ✅ GVAR opcode implementation matches C behavior ⚠️ Zig crashes after 48 instructions (investigation needed)

Next Steps

1. Fix UNBIND crash (Priority 1): Investigate and fix panic in `handleUNBIND` at instruction 3
2. Fix Zig log format: Ensure Zig log format matches C format (complete fields)
3. Fix stack values: Investigate why TOS differs at instruction 2 (GVAR)
4. Re-run comparison: Once fixes are applied, regenerate logs and compare
5. Verify execution: Ensure Zig emulator can execute 1000 instructions without crashing

Related Documentation

- C Emulator PC Advancement Fix - Bug fix details
- Zig GVAR BIGATOMS Implementation - Opcode implementation
- Unified Logging Format - Log format specification