Zig Frame Initialization Analysis

Date: 2026-01-12 16:30 **Status**: In Progress **Purpose**: Understand why Zig initial frame setup differs from C emulator

Problem Statement

The Zig emulator starts execution at PC `0x022400` while the C emulator starts at PC `0x60f130`. The frame information also shows invalid values (0xFFFF) in the Zig emulator log.

C Emulator Frame Setup (Source of Truth)

Key Macros and Definitions

From `maiko/inc/lispemul.h:309`:

```
#define CURRENTFX ((struct frameex1 *)(void *)(((DLword *)PVar) - FRAMESIZE))
```

From `maiko/inc/lispemul.h:479`:

```
#define FRAMESIZE 10 /* size of frameex1: 10 words */
```

From `maiko/src/main.c:819`:

```
PVar = NativeAligned2FromStackOffset(InterfacePage->currentfxp) + FRAMESIZE;
```

Therefore:

- `PVar = Stackspace + currentfxp + FRAMESIZE`
- `CURRENTFX = PVar - FRAMESIZE = Stackspace + currentfxp`

FastRetCALL Macro

From `maiko/inc/retmacro.h:37-45`:

```
#define FastRetCALL \
  do { \
    IVar = NativeAligned2FromStackOffset(GETWORD((DLword *)CURRENTFX - 1)); \
    FuncObj = (struct fnhead *)NativeAligned4FromLAddr(FX_FNHEADER); \
    PC = (ByteCode *)FuncObj + CURRENTFX->pc ; \
  } while (0)
```

Where `FX_FNHEADER` (from `maiko/inc/retmacro.h:28-30`):

- BIGVM: `CURRENTFX->fnheader`
- Non-BIGVM: `(CURRENTFX->hi2fnheader << 16) | CURRENTFX->lofnheader`

C Emulator Execution Flow

From `maiko/src/main.c:771-896`:

1. `start_lisp()` is called
2. `TopOfStack = 0`
3. `PVar = NativeAligned2FromStackOffset(InterfacePage->currentfxp) + FRAMESIZE`
4. `CURRENTFX` is now at `Stackspace + currentfxp`
5. Read `CURRENTFX->nextblock` to set up stack
6. `FastRetCALL` is executed:
   - `FuncObj = NativeAligned4FromLAddr(FX_FNHEADER)`
   - `PC = FuncObj + CURRENTFX->pc`
7. `dispatch()` is called with PC initialized

C Emulator Frame Structure

From `maiko/inc/stack.h:198-220`:

```
typedef struct frameex1 {
  DLword alink;           // offset 0-1
  DLword usecount : 8;    // offset 2 (bits 0-7)
  DLword nopush : 1;      // offset 2 (bit 8)
  DLword validnametable : 1; // offset 2 (bit 9)
  DLword incall : 1;      // offset 2 (bit 10)
  DLword nil2 : 1;        // offset 2 (bit 11)
  DLword fast : 1;        // offset 2 (bit 12)
  DLword flags : 3;       // offset 2 (bits 13-15)

#ifdef BIGVM
  LispPTR fnheader;       // offset 4-7 (32-bit)
#else
  DLword hi2fnheader : 8; // offset 4 (bits 0-7)
  DLword hi1fnheader : 8; // offset 4 (bits 8-15)
  DLword lofnheader;     // offset 6-7
#endif

  DLword pc;              // offset 8-9 (BYTESWAP) or 10-11 (non-BYTESWAP)
  DLword nextblock;       // offset 10-11 (BYTESWAP) or 8-9 (non-BYTESWAP)
  // ... more fields
} frameex1;
```

CRITICAL: The frame structure has different layouts for BYTESWAP vs non-BYTESWAP:

- Non-BYTESWAP: `[4,5]=lofnheader, [6,7]=hi1+hi2, [8,9]=nextblock, [10,11]=pc`
- BYTESWAP: `[4,5]=hi2+hi1, [6,7]=lofnheader, [8,9]=pc, [10,11]=nextblock`

After 32-bit byte-swap (which happens during page loading), the layout becomes:

- `[4,5]=lofnheader, [6,7]=hi1+hi2, [8,9]=pc, [10,11]=nextblock`

Zig Emulator Frame Setup (Current Implementation)

Frame Reading

From `zaiko/src/vm/vm_initialization.zig:40-45`:

```
const STK_OFFSET: u32 = 0x00010000; // DLword offset from Lisp_world
const stackspace_byte_offset = STK_OFFSET * 2; // Convert DLword offset to byte
offset
const currentfxp_stack_offset = ifpage.currentfxp; // DLword offset from Stackspace
const frame_offset = stackspace_byte_offset + (@as(usize,
@intCast(currentfxp_stack_offset)) * 2);
```

This matches C: `CURRENTFX = Stackspace + currentfxp`

Frame Field Reading

From `zaiko/src/vm/vm_initialization.zig:90-141`:

- Reads frame bytes from `virtual_memory[frame_offset..][0..12]`
- Reads `lofnheader` from bytes [4,5] (little-endian)
- Reads `hi1fnheader_hi2fnheader` from bytes [6,7] (little-endian)
- Extracts `hi2fnheader` from low byte of `hi1fnheader_hi2fnheader`
- Calculates `FX_FNHEADER = (hi2fnheader << 16) | lofnheader`
- Reads `pc` from bytes [8,9] (little-endian)
- Reads `nextblock` from bytes [10,11] (little-endian)

Issue: The Zig code assumes BYTESWAP layout after 32-bit swap, but needs to verify this matches C.

PC Calculation

From `zaiko/src/vm/vm_initialization.zig:365-395`:

```
const funcobj_offset_calc: usize = @as(usize, @intCast(fnheader_addr)) * 2; // DLword offset * 2 = byte offset
const frame_pc_bytes = @as(usize, @intCast(frame_pc)); // Use directly as byte offset
const calculated_pc = funcobj_byte_offset + frame_pc_bytes;
```

This matches C: `PC = FuncObj + CURRENTFX->pc`

Debug Output Analysis

C Emulator Debug Output

From C emulator execution log:

- Starting PC: `0x60f130` (Lisp_world+0x60f130)
- Frame: `FX:11890 FH:0x307864 PC:104 NB:11914 FO:+6353096`
- FuncObj offset: `+104 bytes`
- PC = FuncObj + 104 = `0x307864 * 2 + 104 = 0x60f0c8 + 104 = 0x60f130` ✓

Zig Emulator Debug Output

From Zig emulator execution:

- Starting PC: `0x022400` (WRONG!)
- Frame shows: `FX:49408 FH:0xffffff PC:65535 NB:65535 FO:+33554430` (INVALID - 0xFFFF values)

Problem: Zig is reading invalid frame values (0xFFFF), indicating:

1. Frame is not being read correctly, OR
2. Frame is uninitialized/sparse, OR
3. Frame offset calculation is wrong

Findings

Issue 1: Frame Reading

The Zig debug output shows:

```
DEBUG: Reading frame at offset 0x25ce4
First 16 bytes of frame: 0x60 0x2e 0x00 0xc1 0x64 0x78 0x30 0x00 0x68 0x00 0x8a 0x2e ...
```

This frame data looks valid (not all zeros or 0xFF), so the frame is being read.

However, the execution log shows invalid values, suggesting the frame pointer in the VM struct might not be set correctly.

Issue 2: PC Calculation

The Zig code calculates PC correctly from frame, but the frame's `pc` field might be wrong, OR the FuncObj calculation is wrong.

From debug output:

- `FX_FNHEADER = 0x307864` (matches C ✓)
- But PC calculation gives wrong result

Issue 3: Frame Pointer in VM

The Zig code sets `vm.current_frame` at line 178, but the execution trace might be reading from a different location or the frame pointer is not being used correctly.

Root Cause Identified

Issue: Execution Trace Using Wrong Frame Layout

The execution trace in `zaiko/src/vm/execution_trace.zig` was reading the frame as BIGVM (32-bit fnheader at offset 4-7), but the actual frame is non-BIGVM (24-bit fnheader split across bytes 4-7).

Fix Applied (2026-01-12 16:45):

- Changed execution trace to read non-BIGVM frame layout:
  - `lofnheader` from bytes [4,5] (little-endian DLword)
  - `hi1fnheader_hi2fnheader` from bytes [6,7] (little-endian DLword)
  - Extract `hi2fnheader` from low byte (bits 0-7) of `hi1fnheader_hi2fnheader`
  - Calculate `FX_FNHEADER = (hi2fnheader << 16) | lofnheader` (24-bit, masked to 0xFFFFFF)

This matches the frame reading logic in `vm_initialization.zig:90-101`.

Verification

From debug output:

- Zig initialization correctly calculates PC = `0x60f130` ✓
- Dispatch loop starts with correct PC = `0x60f130` ✓
- Execution trace should now show correct frame values (not 0xFFFF)

Next Steps

1. Test execution trace: Run Zig emulator and verify frame values in log match C emulator
2. Fix memory leak: Address the memory leak causing crash after 32 instructions
3. Verify 1000-step log: Ensure Zig emulator generates complete 1000-step log matching C emulator

References

- C source: `maiko/src/main.c:771-896` (start_lisp)
- C macros: `maiko/inc/retmacro.h:37-45` (FastRetCALL)
- C frame struct: `maiko/inc/stack.h:198-220` (frameex1)
- Zig initialization: `zaiko/src/vm/vm_initialization.zig:16-476`
- C emulator log: `c_emulator_execution_log.txt` (first line shows PC=0x60f130)
- Zig emulator log: `zig_emulator_execution_log.txt` (first line shows PC=0x022400)