Comprehensive Bug Analysis with Reasoning (Problems 6-87)

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
P6	3. Omission of Critical Instructions	Fails to calculate the SIMD loop's iteration count.	1	The LLM generated the SIMD loop body but failed to translate the setup logic that determines how many times that loop should run.
P7	4. Incorrect Instruction-Level Semantics	Pattern 4.3: Incorrect Instruction Ordering (resets variable before storing it).	1	The LLM identified the correct instructions but failed to understand their dependency, placing the "reset" action before the "use" action.
P8	1. Incorrect Register State Management	Pattern 1.1: Premature Update (increments index before using it for a store).	1	The LLM failed to recognize that the original value of the index was needed for the str instruction and updated it too early. It didn't use a temporary register.
P9	3. Omission of Critical Instructions	1. Fails to initialize SIMD product vectors. 2. Omits the final SIMD reduction step.	2	The LLM missed both the setup and teardown phases of the SIMD calculation, focusing only on the core loop operation. It's a failure of context.
	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering (overwrites sum vectors during multiplication).	1	The LLM did not allocate enough distinct registers, reusing registers meant for "sum" as destinations for "product", thus corrupting the sum data.
P10	2. Flawed Control Flow and Logic	Pattern 2.2: Premature/Incorrect Return (returns an integer instead of a pointer via a bad branch).	1	The LLM hallucinated an extra basic block and incorrectly redirected control flow, fundamentally misunderstanding the function's intended return value and exit path.
P15	4. Incorrect Instruction-Level Semantics	Pattern 4.1: Invalid Addressing Mode (uses [x0, x0]).	1	The LLM fundamentally misunderstood ARMv8 addressing modes, likely due to a flawed translation from an x86 equivalent where such a pattern might be (differently) valid.
	5. Literal Translation Artifacts (x86-isms)	Mimics a complex shl/sar trick for multiplication instead of	1	The LLM translated the method from x86 (a non-obvious bit-

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
		using a single lsl/sbfiz.		shifting trick) instead of the intent (multiply by 8), failing to use the idiomatic ARMv8 instruction.
P18	1. Incorrect Register State Management	Pattern 1.3: Failure to Propagate Return Values (omits mov x21, x0 after realloc).	1	The LLM did not recognize the convention that x0 holds a function's return value and must be copied to a persistent register before x0 is reused.
	2. Flawed Control Flow and Logic	Pattern 2.1: Incorrect Branch Target (branches to a loop instead of the error handler).	1	The LLM failed to correctly map the if-else structure, connecting the "error" condition to the "success" code path.
	4. Incorrect Instruction-Level Semantics	Pattern 4.2: Incorrect Immediate Value (1. Wrong stack offset. 2. Wrong constant for a note).	2	The LLM either misread or hallucinated the hardcoded constants required for memory addressing and conditional logic.
P19	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering (overwrites the main loop counter with an input value).	1	The LLM failed to track the "liveness" of the loop counter, overwriting it with another value before it was needed, breaking the loop's setup.
P20	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering (overwrites original character with lowercase version before store).	1	The LLM failed to use a temporary register for a calculation, instead modifying the original data in-place and then incorrectly writing the modified data back.
	4. Incorrect Instruction-Level Semantics	Pattern 4.2: Incorrect Immediate Value (uses wrong ASCII values for comparisons).	1	The LLM misidentified or confused the constant values required for character comparisons, breaking the classification logic.
P21	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering (min_diff register s2 is overwritten inside the loop).	1	The LLM failed to track that s2 was a live state variable, reusing it for a temporary load inside the loop and destroying the running minimum.
	4. Incorrect Instruction-Level Semantics	Pattern 4.2: Incorrect Immediate Value (fails to move FLT_MAX into a float register).	1	The LLM correctly identified the constant but failed to generate the instruction to move it from an

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
				integer to a floating-point register for the comparison.
P22	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering (Calculates min/max into s4/s3 but later code incorrectly reads from s0/s1).	1	The LLM failed to maintain a consistent register map. It used one set of registers in the first loop but another set in subsequent code, losing the calculated values.
P23	3. Omission of Critical Instructions	Fails to advance the main string pointer (mov x20, x9) after parsing a number, causing an infinite loop.	1	The LLM missed the crucial state update for the main loop's pointer, causing it to re-process the same data indefinitely.
	1. Incorrect Register State Management	Fails to use the return value from strtol (in w0), storing a garbage character from w8 instead.	1	The LLM didn't know or forgot that w0 holds the return value of strtol, using a different, incorrect register for the result.
P27	1. Incorrect Register State Management	1. Pattern 1.2: Register Clobbering (inner loop counter corrupts outer loop's index). 2. Stores a loop counter instead of an array element.	2	The LLM failed to manage nested loop state, reusing the outer loop's index register for the inner loop. It also confused a counter with a data value.
	3. Omission of Critical Instructions	Fails to initialize a separate counter for an inner loop.	1	The LLM didn't generate the setup code for a nested loop, which led to the register clobbering error.
P28	7. Misinterpretation of Algorithm's Goal	Grossly incorrect SIMD implementation (unrolls to 64 bytes and botches logic).	1	The LLM failed to understand the compact SIMD logic, likely misinterpreting x86, and attempted to compensate with a massive, incorrect unrolling.
	3. Omission of Critical Instructions	Completely omits the logic for swapping uppercase to lowercase in the scalar path.	1	The LLM missed one entire branch of an if-else if-else structure, generating code for only one of the two main cases.
P29	1. Incorrect Register State Management	Fails to preserve the array pointer for a second pass, consuming it in the first loop.	1	The LLM didn't recognize that the array pointer was needed twice and failed to create a temporary copy for the first loop, thus modifying the original.

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
P33	1. Incorrect Register State Management	1. Fails to update the main state variable (d0). 2. Calculates powers of the wrong variable.	2	The LLM failed to identify the key state variable (d0) of the algorithm, consistently using and updating the wrong registers.
	2. Flawed Control Flow and Logic	Incorrect branching; falls through into hallucinated code instead of looping correctly.	1	The LLM lost the program's structure, failing to generate correct loop-back branches and creating dead/spurious code blocks.
	9. Code Hallucination	Generates several blocks of nonsensical, spurious code.	1	The LLM completely failed to comprehend the algorithm and generated syntactically valid but semantically meaningless instructions, a severe model failure.
P35	1. Incorrect Register State Management	Initializes an inner loop counter with the array element's value instead of the count of items to check.	1	The LLM confused a data value with a loop bound, a fundamental logic error in setting up a nested loop.
	3. Omission of Critical Instructions	Completely omits the function epilogue (updating out_count, restoring registers, ret).	1	The LLM generated the main body but failed to generate the required function exit code, creating an incomplete and non-functional result.
P36	1. Incorrect Register State Management	Resets the max_val register back to its initial small value in every loop iteration.	1	The LLM failed to understand the concept of a "running total" or "running max," incorrectly placing an initialization instruction inside the loop instead of before it.
P37	7. Misinterpretation of Algorithm's Goal	Fails to translate compound conditional logic, incorrectly merging two independent checks with ccmp.	1	The LLM misidentified two separate if conditions as a single, dependent one, showing a failure to understand the high-level logic of the x86 branching.
	3. Omission of Critical Instructions	Omits a key mul instruction needed for a conditional check.	1	The LLM failed to generate one of the necessary calculations for the compound condition, making the

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
				check impossible to perform correctly.
P38	6. Incorrect Loop Pointer/Index Management	Fails to advance the main SIMD source pointer, causing an infinite loop on the same data.	1	The LLM missed the pointer update instruction for the main loop, causing it to process the same data block repeatedly.
	3. Omission of Critical Instructions	Completely omits the final merging loop that constructs the output array.	1	The LLM generated code for the intermediate steps but failed to generate the final, crucial stage of the algorithm that produces the output.
	2. Flawed Control Flow and Logic	Pattern 2.1: Incorrect Branch Target (branches out of an inner sort loop prematurely).	1	The LLM generated the wrong branch target inside a standard algorithm (Insertion Sort), breaking its fundamental logic.
P40	1. Incorrect Register State Management	Operates on a copy of the main state variable (n), which is never updated inside the inner loop.	1	The LLM failed to understand that the state variable n needed to be modified in-place, instead operating on a copy and thus losing all updates.
	2. Flawed Control Flow and Logic	Unconditional branch in a nested loop creates an infinite loop.	1	The LLM replaced a conditional loop-back with an unconditional one, showing a failure to translate the loop's termination condition.
P41	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering (Completely corrupts all loop iterators and flags at setup).	1	The LLM failed to manage multiple state variables, creating a sequence of mov instructions that overwrote its own iterators before they could be used.
	7. Misinterpretation of Algorithm's Goal	Fails to understand the j+k < size loop condition, checking k < size instead.	1	The LLM missed the relational dependency between nested loop counters, simplifying the loop bound check in a way that breaks the algorithm's correctness.
	2. Flawed Control Flow and Logic	Pattern 2.2: Premature/Incorrect Return (exits immediately on first match instead of continuing search).	1	The LLM incorrectly assumed the goal was to find the first match and return, when the original logic required checking all possibilities.

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
P44	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering (Completely corrupts loop iterators and flags at setup, similar to P41).	1	Same as P41; a catastrophic failure of register management at the start of the loops, where state variables are overwritten before being used.
	2. Flawed Control Flow and Logic	Pattern 2.2: Premature/Incorrect Return (exits immediately on first match).	1	Same as P41; the LLM misunderstood the algorithm's goal and generated an incorrect early exit.
P45	6. Incorrect Loop Pointer/Index Management	Uses a stagnant (never incremented) write pointer, overwriting out_str[0] in every loop iteration.	1	The LLM generated the loop body but forgot to include the instruction to advance the destination pointer, a critical part of any copy loop.
	8. Failure to Generate Idiomatic/Optimized Code	Fails to generate the optimized SIMD path (rev64.8b) for string reversal.	1	The LLM failed to recognize the opportunity for a high-performance, target-specific optimization, falling back to a (buggy) scalar version.
P47	6. Incorrect Loop Pointer/Index Management	Instruction Reordering: The relative order of ldr and pointer updates is wrong, causing the recurrence relation to use stale data.	1	The LLM generated the correct instructions but in the wrong sequence, failing to understand the data dependency between loop iterations in a recurrence relation.
P50	1. Incorrect Register State Management	Pattern 1.4: Incorrect Source/Destination in Update: Fails to use the running result as input for the next iteration.	1	The LLM broke the chain of the recurrence relation, using a static initial value as input for each step instead of the result from the previous step.
P51	9. Code Hallucination	Generates a nonsensical, 500+ instruction block instead of a simple SIMD loop.	1	The LLM encountered a complex pattern it didn't understand (a SIMD loop with a table lookup) and generated a massive, meaningless block of code. A total model failure.
	3. Omission of Critical Instructions	Completely omits the scalar fallback path for short strings.	1	The LLM failed to generate code for one of the main if-else branches of the algorithm, leaving

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
				it unable to handle a large class of inputs.
P60	1. Incorrect Register State Management	Operates on a temporary copy of n inside the "divide out" loop, so n is never updated.	1	Same as P40; a failure to modify a key state variable in-place, leading to an infinite loop because the loop-breaking condition is never met.
	2. Flawed Control Flow and Logic	Unconditional branch in the "divide out" loop creates an infinite loop.	1	Same as P40; the LLM failed to generate the conditional exit for a nested loop.
P63	4. Incorrect Instruction-Level Semantics	Pattern 4.1: Invalid Addressing Mode: Provides a pre-scaled offset where a raw index is expected, causing double scaling.	1	The LLM misunderstood the target's specific addressing modes, manually scaling an index and then providing it to an instruction that performs its own scaling.
P64	4. Incorrect Memory Addressing and Management	Uses incorrect stack offsets for its temporary array.	1	The LLM made a simple but critical error in calculating a memory offset on the stack, risking a buffer overflow or data corruption.
	6. Incorrect Loop Pointer/Index Management	str instruction corrupts the base pointer used for ldur in the next iteration.	1	The LLM generated an instruction that advanced a pointer that was supposed to remain static for reading, breaking the data access for the next loop iteration.
	1. Incorrect Register State Management	Fails to preserve the input n, using an uninitialized register for the final read index.	1	The LLM failed to save a value that was needed much later, using a register with junk data for the final operation.
P65	1. Incorrect Register State Management	Consumes the main string pointer in the first loop, so it's invalid for the second operation (reading the last character).	1	Same as P29; a failure to recognize that a pointer was needed for two separate passes, causing it to be consumed by the first pass.
	7. Misinterpretation of Algorithm's Goal	Incorrectly translates a simple if...then into a flawed csinc instruction.	1	The LLM chose a complex instruction to implement simple logic but configured it with the

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
				wrong condition and destination, showing it didn't understand the goal.
P66	4. Incorrect Memory Addressing and Management	Uses an incorrect and dangerously small stack offset for a temporary buffer, risking a buffer overflow.	1	Similar to P64; an error in stack offset calculation that creates a severe memory safety risk.
	2. Flawed Control Flow and Logic	Pattern 2.1/2.2: Incorrect branching after main logic fails to set the return value correctly for one code path.	1	The LLM failed to correctly structure the program's control flow graph, leaving one of the main logic paths unable to reach the common function exit.
P68	7. Misinterpretation of Algorithm's Goal	Fails to translate a compound conditional for state switching, producing jumbled ccmp/csel logic.	1	Similar to P77; the LLM saw multiple conditions and incorrectly tried to merge them into a single dependent check, failing to grasp the high-level logic.
	1. Incorrect Register State Management	Uses the wrong length register (len1) as the index into the second number's buffer.	1	The LLM confused two different state variables (len1 and len2), using the wrong one in a critical memory write operation.
P71	6. Incorrect Loop Pointer/Index Management	Instruction Reordering: Decrements a "from-end" index before the ldr, causing an off-by-one read.	1	The LLM generated the correct instructions but in the wrong order, causing an off-by-one error in data access. A failure to understand data dependency.
	1. Incorrect Register State Management	Pattern 1.1: Premature Update: Increments the output write index before all writes for the current iteration are complete.	1	The LLM updated a state variable too early, before its original value was used for a second required operation in the same loop iteration.
P72	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering: Destroys input registers (a, b, c) during the Triangle Inequality check.	1	The LLM failed to use temporary registers for intermediate calculations, instead overwriting its own input values and corrupting the state needed for later calculations.

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
	7. Misinterpretation of Algorithm's Goal	Fails to understand that the inequality check requires all original values, leading to the state destruction.	1	The LLM didn't grasp the high-level requirement of the validation algorithm, which led directly to the register clobbering error.
P76	1. Incorrect Register State Management	Operates on a temporary copy of n inside the "divide out" loop, so n is never updated. (Same as P60).	1	Same as P60; a failure to modify a key state variable in-place, leading to an infinite loop because the loop-breaking condition is never met.
	2. Flawed Control Flow and Logic	Unconditional branch in the "divide out" loop creates an infinite loop. (Same as P60).	1	Same as P60; the LLM failed to generate the conditional exit for a nested loop.
P77	7. Misinterpretation of Algorithm's Goal	Incorrectly merges two independent loop conditions (power > n and count < 100) into a single faulty ccmp.	1	Same as P37; a failure to understand that two conditions were independent, leading to an incorrect merging of logic with ccmp.
	1. Incorrect Register State Management	Unconditionally increments the loop counter, even on the final iteration that exits the loop.	1	The LLM placed a state update (count++) in a position where it executes one too many times, a subtle off-by-one logic error.
P81	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering: The fixed pattern character register is overwritten inside the loop.	1	The LLM failed to recognize that a register held a constant value needed for every loop iteration, allowing it to be overwritten and destroying the pattern.
	7. Misinterpretation of Algorithm's Goal	Completely misinterprets the loop's comparison logic.	1	The LLM failed to understand the goal of the loop ("compare against fixed pattern") and instead generated code that compared against the previous character.
P82	4. Incorrect Instruction-Level Semantics	Pattern 4.2: Incorrect floating-point constants. Pattern 4.1: Invalid addressing mode (str x0, [x0]).	2	The LLM generated incorrect hardcoded data and also failed to understand the basic rules of ARMv8 memory addressing.
	6. Incorrect Loop Pointer/Index Management	Uses a stagnant output pointer, writing every result to out_array[0].	1	Same as P45; the LLM forgot to include the instruction to advance

Problem	Category	Pattern / Specific Error	Freq.	Reasoning for Failure
				the destination pointer inside the loop.
P85	6. Incorrect Loop Pointer/Index Management	Uses incorrect pointer arithmetic to advance through a string.	1	The LLM generated a nonsensical sequence of instructions to advance a pointer instead of a simple increment, showing confusion about pointer arithmetic.
	2. Flawed Control Flow and Logic	Uses an incorrect termination condition for a string reversal loop.	1	The LLM failed to correctly translate the loop's for (i=0; i < len/2; ++i)-style termination condition.
P86	1. Incorrect Register State Management	Pattern 1.2: Register Clobbering: A register holding a loop index is overwritten by a SIMD result before it's used.	1	The LLM reused a register for two different critical purposes without saving the first value, destroying the index needed for the scalar loop.
P87	1. Incorrect Register State Management	1. Fails to initialize a pointer for the inner sort loop. 2. Pattern 1.1: Premature Update (resets word length before use).	2	The LLM failed to initialize a critical pointer and also placed a state-resetting instruction before the state was fully used, showing multiple levels of state confusion.

Analysis Summary

Key Insights from Reasoning Patterns

1. Register State Management Failures (Category 1)

- **Register Clobbering (Pattern 1.2)**: Most common failure mode - LLMs consistently fail to track register liveness and reuse registers inappropriately
- **Premature Updates (Pattern 1.1)**: LLMs update state variables before all uses of the original value are complete
- **Failure to Propagate Return Values (Pattern 1.3)**: Missing understanding of calling conventions

2. Algorithm Comprehension Issues (Category 7)

- LLMs struggle with compound conditional logic, often incorrectly merging independent conditions
- Failure to understand high-level algorithm goals leads to cascading errors
- Particular difficulty with state machine logic and complex control structures

3. Context and Dependency Failures

- LLMs miss critical setup/teardown phases of algorithms (especially in SIMD code)
- Poor understanding of data dependencies between instructions
- Instruction reordering errors that break recurrence relations

4. Code Generation Pathologies

- **Code Hallucination (Category 9)**: Complete model failures resulting in massive blocks of nonsensical code
- **x86-isms (Category 5)**: Literal translation artifacts from x86 patterns instead of understanding intent
- Missing optimization opportunities and idiomatic code patterns

5. Loop and Pointer Management

- Systematic failures in advancing pointers within loops
- Off-by-one errors due to incorrect instruction ordering
- Infinite loops from missing state updates

Recommendations for Model Improvement

1. **Enhanced Register Allocation Training**: Focus on register liveness analysis and temporary register usage
2. **Control Flow Graph Understanding**: Better training on complex conditional logic and state machines
3. **Architecture-Specific Training**: Reduce x86 translation artifacts through ARM-native training data
4. **Dependency Analysis**: Improved understanding of instruction dependencies and ordering constraints
5. **Algorithm Pattern Recognition**: Better recognition of common algorithmic patterns and their implementations