

| Entries | execute | interpreter | journal&state |
|---|--|---|---|
| main revm/examples/uniswap_get_reserves/src/main.rs | call revm/crates/revm/src/handler/handle_types/execution.rs | | |
| let mut cache_db = CacheDB::new(EmptyDB::default()); let mut evm = Evm:: <ethereumwiring<cachedb<emptydb>, ()>>::builder() let ref_tx = evm.transact().unwrap();</ethereumwiring<cachedb<emptydb> | (self.call)(context, inputs) Call Freym/crates/revm/src/handler/mainnet/execution.rs | | |
| transact revm/crates/revm/src/evm.rs | context.evm.make_call_frame(&inputs) | | |
| let output = self.transact_preverified_inner(initial_gas_spend); let output = self.handler.post_execution().end | make_call_frame revm/crates/revm/src/context/evm_contetxt.rs | | |
| transact_preverified_inner revm/crates/revm/src/evm.rs | CallValue::Transfer(value) if value.is_zero() => { self.load_accountf(inputs.target_address) self.journaled_state.touch(&inputs.target_address); | | load_account revm/crates/revm/journaled_state.rs |
| let exec = self.handler.execution(); let first_frame_or_cation = exec.first_frame_creation(ctx, gas_limit)?; let first_frame_or_result = match first_frame_action { NewFrameAction::Call(inputs) => exec.call(ctx, inputs)?, NewFrameAction::Creatle (inputs) => exec.certecte(ctx, inputs)?, NewFrameAction::Creatle (inputs) => exec.certecte(ctx, inputs)?, FrameOrResult::Frame(first_frame) => self.run_the_loop(first_frame)?, run_the_loop revm/crates/revm/src/evm.rs | Ok(FrameOrResult::new_call_frame(| run revm/crates/interpreter/src/interpreter.rs | let load = match self.state.entry(address) { Entry::Occupied(entry) => { StateLoad { Entry::Vacant(vac) => { et account = if let Some(account) = db.basic(address)? { } |
| | execute_frame revm/crates/revm/src/handler/handle_types/execution.rs | while self.instruction_result == InstructionResult::Continue { | |
| | (self.execute_frame) (frame, shared_memory, instruction_tables, context) | step revm/crates/interpreter/src/interpreter.rs let opcode = unsafe { "self.instruction_pointer }; | |
| | execute_frame revm/crates/revm/src/handler/mainnet/execution.rs | | |
| loop { | InstructionTables::Plain(table) => interpreter.run(memory, table, context). | self.instruction_pointer = unsafe { self.instruction_pointer.offset(1) }; (instruction_table(opcode as usize))(self, host) | |
| | | sload revm/crates/interpreter/src/interpreter.rs | |
| | | let Some(value) = host.sload(interpreter.contract.target_address, | |
| | sload revm/crates/revm/src/context.rs | | |
| | self.evm.sload(address, index) | | |
| | sload revm/crates/revm/src/context/inner_evm_context.rs | | |
| | self.journaled_state.sstore(address, index, value, &mut self.db) | | sload revm/crates/revm/journaled_state.rs |
| | | | db.storage(address, key)? |
| | | | |
| | | | |