AetherRISC: Engineering Project Plan & Roadmap

Project: AetherRISC

Version: 0.0.0 (Pre-Alpha)

Date: 2025-12-21

Standard Compliance: IEEE 12207 (Software Life Cycle Processes)

Stage 1: Requirements Analysis & Architectural Definition

Objective: Establish the foundational software architecture, define rigid API contracts, and freeze data schemas to ensure modularity across the lifecycle.

- **Task 1.1: System Architecture Definition**
  - Define the Solution Topology (AetherRISC.Core, AetherRISC.UI, AetherRISC.Web, AetherRISC.Desktop).
  - Establish the Dependency Injection (DI) strategy for Hybrid (Server/WASM) execution.
- **Task 1.2: API Contract Definition**
  - Define IInstruction, IPipelineStage, and IMemoryBus interfaces.
  - Define the IInstructionSetExtension plugin interface for dynamic ISA loading.
- **Task 1.3: Data Schema Finalization**
  - Define SystemConfig JSON schema for hardware definition.
  - Define Google Protobuf contracts (MachineStateSnapshot) for client-server state synchronization.
- **Deliverable:** System Architecture Document (SAD) v1.0, Frozen Interface Definitions.

Stage 2: Core Kernel Implementation (The "Brain")

Objective: Implement the platform-agnostic simulation logic, focusing on correctness and Venus/RARS legacy parity.

- **Task 2.1: Memory Subsystem**
  - Implement PhysicalMemory with Endianness handling.
  - Implement MemoryBus routing logic.
- **Task 2.2: Register Architecture**
  - Implement GeneralPurposeRegisterFile (x0-x31).
  - Implement ControlStatusRegisterBank (CSRs) with privilege level logic.
- **Task 2.3: Base ISA Implementation**

    - Implement InstructionDecoder using Source Generators for performance.
    - Implement RV32I / RV64I Base Integer Instruction Sets.
- **Task 2.4: Legacy Compatibility Layer**
    - Implement SyscallDispatcher complying with Venus/RARS IDs (1-57).
    - Implement ElfReader and VenusProjectReader for asset ingestion.
- **Deliverable:** AetherRISC.Core.dll passing 100% of Unit Tests for arithmetic and memory operations.

Stage 3: Infrastructure & Presentation Layer

Objective: Establish the hosting environments and the user interface shell.

- **Task 3.1: Hosting Infrastructure**
    - Configure AetherRISC.Web with Blazor InteractiveAuto (Server + WASM).
    - Configure AetherRISC.Desktop (.NET MAUI) with WebView2 bridging.
- **Task 3.2: Simulation Service Bridge**
    - Implement SimulationRunner service (Scoped) for client-side loop management.
    - Implement State Synchronization logic (Protobuf serialization).
- **Task 3.3: Editor Integration**
    - Implement MonacoWrapper.razor with custom RISC-V syntax highlighting.
    - Implement ReferencePane.razor populated from reference.json (Venus Parity).
- **Deliverable:** Functional UI Shell capable of loading, assembling, and stepping through code.

Stage 4: Micro-Architectural Simulation

Objective: Implement the cycle-accurate features that distinguish AetherRISC from basic emulators.

- **Task 4.1: Pipeline Logic**
    - Implement PipelineController and the 5-Stage FSM (IF, ID, EX, MEM, WB).
    - Implement PipelineLatch data structures.
- **Task 4.2: Hazard Management**
    - Implement HazardDetectionUnit (Stall injection logic).
    - Implement ForwardingUnit (Data bypassing logic).
- **Task 4.3: Memory Hierarchy**
    - Implement CacheController logic (Write-Back/Write-Allocate).
    - Implement L1 (Split I/D) and L2 (Unified) cache arrays.
    - Implement CacheGrid.razor visualizer.
- **Deliverable:** Fully visualized pipeline with stall/hazard indicators and cache hit/miss animation.

Stage 5: Runtime Intervention & Diagnostics (RIS/HDE)

Objective: Implement the "Pro" features for analysis and intervention.

- **Task 5.1: Runtime Intervention Subsystem (RIS)**
  - Implement InterventionManager facade.
  - Implement MemoryPatcher for Dynamic Code Injection (DCI/Trampolining).
  - Implement DirectRegisterAccess (DRA) with pipeline flushing logic.
- **Task 5.2: Heuristic Diagnostic Engine (HDE)**
  - Implement StaticAnalyzer for pre-flight checks.
  - Implement StallExplainer for deterministic hazard messaging.
  - Implement DecompilerService (ASM-to-Pseudo-C projection).
- **Task 5.3: Advanced Visualization**
  - Implement SystemControlPanel.razor (formerly God Mode UI).
  - Implement RegisterBitInspector.razor.
- **Deliverable:** Working "God Mode" features and deterministic explanation engine.

Stage 6: Embedded Systems & External Interfaces

Objective: Expand the simulation to include peripheral devices and external tool connectivity.

- **Task 6.1: Embedded IO Subsystem**
  - Implement MmioBus and IMmioDevice interface.
  - Implement GpioController and UartController.
  - Implement PcbView.razor (SVG binding).
- **Task 6.2: Polyglot Compilation Service**
  - Implement Docker container orchestration for GCC/Rust toolchains.
  - Implement DWARF symbol parsing for source-level debugging.
- **Task 6.3: External Debugging**
  - Implement GDB Remote Serial Protocol (RSP) Stub (TCP Listener).
- **Deliverable:** End-to-end "Blinky" demo on Virtual PCB and GDB attachment capability.

Stage 7: Verification, Validation & Deployment

Objective: Ensure rigorous standard compliance and system stability.

- **Task 7.1: Architectural Verification**
  - Integrate RISCOF (RISC-V Architectural Test Framework).
  - Validate against Golden Reference Models (Spike/Sail).
- **Task 7.2: Regression Testing**
  - Execute Venus and RARS test suites to guarantee legacy parity.
- **Task 7.3: Documentation & Release**
  - Generate User Manual and Migration Guide.
  - Publish Docker Images and NuGet Packages.
- **Deliverable:** Release Candidate 1.0 (Gold Master).