Section 5: Kernel Stub & Early Boot

Versioned Specifications (v1–v5)

Computer Ecosystem Project

January 29, 2026

Contents

1 Overview

This document aggregates Section 5 (Kernel Stub & Early Boot) specifications across versions **v1**–**v5**. It includes:

- a comparison table showing what each version adds; and
- the full specification text for each version using `\input`.

2 Version Comparison Table (v1–v5)

2.1 High-level feature comparison

Capability / Rule	v1	v2	v3	v4	v5
Target CPU baseline	CPU v2	CPU v2	CPU v3	CPU v4	CPU v5
Output format	Flat `.bin`	Flat `.bin`	Image-mode (`vectors` + `entry`)	Image-mode + `.data`/fixed BootInfo	Dual-image (`kernel.bin` + `user.bin`)
Reset entry PC	`0x0000`	`0x0000`	`0x0100`	`0x0100`	`0x0100`
Vector table (0x0000..0x00FF)	— (not used)	— (not used)	Required	Required	Required + protection vector
Kernel stub responsibilities	Stack init + jump	+ sanity checks + panic convention	+ syscall/-timer entry paths	+ Boot ABI + BootInfo + v4 alignment discipline	+ kernel→user transition + protection boundary
Deterministic panic signature	Optional (minimal)	Required (R0..R4)	Required (vector-aware)	Required (BootInfo-aware)	Required (adds prot-fault signature)
Syscalls	No	No	Yes (SYSCALL via vector 0x00)	Yes (ABI stabilized + preservation rules)	Yes (user uses syscalls; kernel enforces boundary)
Timer interrupt	No	No	Yes (vector 0x01 minimal handler)	Yes	Yes
Boot ABI into `kernel_main`	Implicit	Implicit	Minimal (stack + jump)	**Frozen**: `R0=bootinfo_ptr`	Frozen + extended BootInfo for user image
BootInfo block	No	No	Optional (debug-only)	**Required** (v4 layout)	**Required** (v5 extended layout)
Protection / user mode	No	No	No	No	**Yes** (K flag + traps + dual-image)

2.2 Compatibility summary (what stays invariant)

- Instruction encoding remains fixed (8 bytes) across versions.
- v3+ reserves vectors at `0x0000..0x00FF` and resets at `0x0100`.
- Stack remains below reserved MMIO start `0xFE00`, with recommended kernel stack top `0xFDFF`.
- In v5, user-mode may access exttt0x0100..0xFDFF; only vectors (exttt0x0000..0x00FF) and

MMIO/reserved (exttt0xFE00..0xFFFF) are privileged. exttt0xF800..0xFDFF may be used by the kernel stack by convention.

- Syscall entry vector remains `0x00` from v3 onward.

3 Specifications by Version

4 Section 5 v1: Kernel Stub & Early Boot (Minimal Boot Bridge)

Purpose

A **kernel stub** is the first OS-controlled code executed after CPU reset. In this project its role is intentionally narrow:

- establish the minimum execution environment that higher-level kernel code assumes;
- transfer control deterministically to the kernel entry function;
- remain forward-compatible with later CPU/toolchain versions.

Non-goal: the stub is not "the kernel"; it is the bridge that makes kernel code possible.

Version intent

v1 is the most minimalistic version that still works for this project: initialize a valid stack and jump to `kernel_main`. Everything else (diagnostics, vectors, syscalls, protection) is deferred to later versions.

4.1 Compatibility Contract (from earlier sections)

- **CPU baseline: v2.** v2 is the earliest version that makes `SP`/`FP` meaningful for a real stub.
- **Instruction encoding fixed at 8 bytes** across all versions (forward-compatibility invariant).
- **Address space:** 64KB RAM, byte-addressed, `0x0000..0xFFFF`.
- **Little-endian** (relevant for call/return conventions later).
- **Assembler baseline: v2**, output is a **flat binary** `.bin` loaded at `0x0000`.

4.2 Reset and entry assumptions (v1)

The CPU v2 reset convention provides a deterministic baseline:

- **PC** starts at `0x0000`.
- **GPRs** start at 0; **Z** starts cleared.
- **SP** and **FP** start at `0xFDFF`.

v1 still explicitly sets `SP` and `FP` to avoid depending on reset defaults.

4.3 Scope

In scope (required):

1. Initialize stack (`SP`) and frame pointer (`FP`).
2. Transfer control to `kernel_main`.

Out of scope: vectors, syscalls, interrupts, MMIO, allocators, protection/user mode.

4.4 Memory Plan (v1 recommended)

Region	Address range	Usage
Kernel image (flat)	`0x0000 .. <end>`	Stub + early kernel code
Free RAM	`<end> .. 0xF7FF`	Unused in v1
Kernel stack	`0xF800 .. 0xFDFF`	Downward-growing; top at `0xFDFF`
Reserved future region	`0xFE00 .. 0xFFFF`	Keep unused (future MMIO)

4.5 Control Flow (normative)

```
_start (PC=0x0000):
  SP := 0xFDFF
  FP := 0xFDFF
  JMP_ABS kernel_main
```

4.6 Kernel entry contract (v1 ABI)

On entry to `kernel_main`:

- `SP` points to the stack-top (recommended `0xFDFF`);
- `FP = SP`;
- no interrupts, syscalls, or IO are assumed.

4.7 Plan of Execution (v1)

Deliverables:

- `kstub_v1.asm` implementing stack init + jump;
- `kernel_main.asm` proving control transfer (deterministic `HALT` is enough);
- flat-binary build using assembler v2.

4.8 Definition of Done (v1)

v1 is complete when reset always reaches `kernel_main` and ends deterministically.

5 Section 5 v2: Kernel Stub & Early Boot (Deterministic Diagnostics)

Purpose

v2 preserves v1's minimal boot bridge (stack init + jump) and adds a frozen, deterministic early-boot diagnostics model:

If early boot fails, it fails loudly and repeatably via a panic signature.

5.1 Compatibility Contract

- **CPU baseline: v2** (stack model exists).
- **Assembler baseline: v2**, output stays flat `.bin` loaded at `0x0000`.
- No vectors, syscalls, or MMIO are used in v2.

5.2 Scope

Adds to v1:

- boot stage identifiers;
- minimal sanity checks;
- a frozen panic contract (register signature + panic codes).

5.3 Boot stages (frozen IDs)

Stage ID	Meaning
`0x01`	Entered `_start`
`0x02`	Stack initialized
`0x03`	Sanity checks running
`0x04`	Transfer to `kernel_main` imminent

5.4 Panic contract (normative)

On unrecoverable early-boot failure, the stub must set and then halt:

Register	Meaning
`R0`	Panic code
`R1`	Stage ID
`R2`	Location identifier (PC snapshot if available, else label constant)
`R3`	SP snapshot
`R4`	FP snapshot

5.5 Panic codes (normative)

Code	Meaning
`0x10`	SP out of allowed range
`0x11`	SP overlaps reserved future region (`0xFE00..0xFFFF`)
`0x12`	SP overlaps kernel code region (unsafe)
`0x13`	Alignment violation (PC/target not 8-byte aligned)
`0x14`	Kernel entry invalid/out of bounds

5.6 Sanity checks (mandatory)

Before jumping to `kernel_main`, v2 must verify:

1. SP is within `[STACK_BOTTOM..STACK_TOP]` (recommended `0xF800..0xFDFF`);
2. SP is below `0xFE00` (reserved for future MMIO);
3. `kernel_main` is valid and 8-byte aligned.

5.7 Control flow (normative)

```
_start:
  stage := 0x01
  SP := STACK_TOP
  FP := STACK_TOP
  stage := 0x02

  stage := 0x03
  run sanity checks
  if fail -> PANIC(code, stage, location)

  stage := 0x04
  JMP_ABS kernel_main
```

5.8 Definition of Done (v2)

- normal boot reaches `kernel_main` exactly like v1;
- at least one deliberate misconfiguration triggers a correct panic signature.

6 Section 5 v3: Kernel Stub & Early Boot (Vectors + Reset @ 0x0100)

Purpose

v3 transitions the project into an OS-shaped boot model:

- vector table at `0x0000..0x00FF`;
- reset entry at `PC=0x0100`;
- `SYSCALL` enters vector 0x00 and returns via `IRET`;
- timer interrupt enters vector 0x01 and returns via `IRET`.

6.1 Compatibility Contract

- **CPU baseline: v3**.
- **Toolchain baseline: v3 image-mode** capable of placing vectors and a fixed entry.
- Vector entries are **u16 little-endian addresses** at `0x0000 + 2*id`.
- Reset begins at `0x0100` (not `0x0000`).

6.2 Scope

Adds to v2:

- required vector table and handlers;
- syscall entry/return proof-of-life;
- timer IRQ entry/return proof-of-life.

Still out of scope: protection/user mode and dual-image boot.

6.3 Memory and image layout (normative)

Region	Range	Rule
Vectors	`0x0000..0x00FF`	Present; kernel-owned
Reset entry	`0x0100`	`_start` placed here
Reserved MMIO	`0xFE00..0xFFFF`	Reserved for devices

6.4 Required vectors (v3)

ID	Handler	Requirement
`0x00`	`syscall_handler`	Must end with `IRET`
`0x01`	`timer_handler`	Must end with `IRET`

6.5 Minimal syscall ABI (v3)

- `R0`: syscall number / return value
- `R1..R3`: up to three arguments

Minimal proof-of-life syscalls:

- `1 = sys_putc` (write `R1` to MMIO console or debug RAM);
- `2 = sys_halt`.

6.6 Interrupt model (v3/v4)

v3 introduces asynchronous interrupts (used by the timer IRQ and optionally other future IRQs). **There is no architectural interrupt-enable/mask bit in v3/v4.** In other words, software cannot selectively disable interrupts at the ISA level until v5.

Normative entry/return rules:

- On IRQ entry, the CPU transfers control to the handler address read from the vector table (e.g., timer IRQ uses vector `0x01`).
- The return address is saved in `EPC` (per the CPU roadmap), and the handler returns with `IRET`, restoring `PC` from `EPC`.
- Any additional trap metadata (e.g., `CAUSE`, `BADADDR`) is handled as defined by the CPU roadmap for the target version.

6.7 Boot stages and diagnostics continuity (normative)

For deterministic diagnostics, v3 **inherits the v2 boot stage IDs unchanged** for use in the panic signature. Implementations may report the same stage values as v2 (e.g., 0x01..0x04) when failing during early boot.

6.8 Boot flow (normative)

```
Reset:
  PC := 0x0100

_start @ 0x0100:
  SP := 0xFDFF
  FP := 0xFDFF
  JMP_ABS kernel_main
```

6.9 v3 panic signature (normative)

On unrecoverable fault, set and halt:

Reg	Meaning
`R0`	Panic code
`R1`	Stage ID
`R2`	Location identifier (EPC snapshot if available, else constant/label)
`R3`	SP snapshot
`R4`	FP snapshot

6.10 Definition of Done (v3)

- image boots at `0x0100`;
- `SYSCALL` enters vector 0x00 and returns via `IRET`;
- timer IRQ enters vector 0x01 and returns via `IRET`.

7 Section 5 v4: Kernel Stub & Early Boot (Boot ABI + BootInfo)

Purpose

v4 keeps v3's layout and vectors unchanged, but stabilizes early boot as a platform for real kernel development by introducing:

- a frozen **Boot ABI** into `kernel_main`;
- a structured **BootInfo** block passed to the kernel;
- v4 alignment discipline enabling safe 16/32/64-bit memory operations and base+offset addressing.

7.1 Compatibility Contract

- **CPU baseline: v4**.
- v3 vector layout is preserved (vectors at `0x0000..0x00FF`, reset `0x0100`).
- Toolchain supports image-mode plus `.data` (or fixed placement) for BootInfo.

7.2 Boot ABI (normative)

Kernel entry is:

```
kernel_main(bootinfo_ptr)
```

Passing convention:

- `R0 = bootinfo_ptr`;
- `SP` initialized to kernel stack top (recommended `0xFDFF`);
- `FP = SP`;

7.3 BootInfo placement (v4 new rule)

- **Recommended fixed address:** `BOOTINFO_ADDR = 0x0200`.
- **No-overlap rule (normative):** the kernel image (code/data/bss/stack init) must not overwrite the BootInfo block: reserve the range `[BOOTINFO_ADDR .. BOOTINFO_ADDR + size)` as kernel-owned metadata. If a fixed address is used, the linker/loader must place kernel sections so they do not overlap this range.
- Must be at least 8-byte aligned.
- Multi-byte fields are little-endian and naturally aligned.

7.4 BootInfo v4 layout (normative)

Offset	Type	Name	Meaning
0x00	u16	`magic`	`0xB007`
0x02	u8	`version`	4
0x03	u8	`size`	total size (recommended 0x20)
0x04	u16	`mem_size`	65536
0x06	u16	`flags`	feature flags
0x08	u16	`kernel_entry`	0x0100
0x0A	u16	`vector_base`	0x0000
0x0C	u16	`stack_top`	0xFDFF
0x0E	u16	`stack_bottom`	0xF800
0x10	u16	`reserved_mmio`	0xFE00
0x12	u16	`dbg_out`	optional debug RAM addr
0x14	u32	`boot_time`	optional (0 if unused)
0x18	u32	`reserved0`	reserved

0x1C	u32	`reserved1`	reserved

7.5 Alignment and v4 memory ops (normative)

- 16-bit ops: addr mod 2 = 0; 32-bit: mod 4 = 0; 64-bit: mod 8 = 0.
- v4 boot must include a minimal proof routine that:
  - reads BootInfo fields using base+offset addressing, and
  - clears a small aligned region using 64-bit stores.

7.6 Syscall ABI refinement (v4)

Syscall register convention remains: `R0` syscall #, `R1..R3` args, `R0` return. v4 additionally freezes that syscall handlers must preserve callee-saved registers.

7.7 Definition of Done (v4)

- `kernel_main` receives `R0=0x0200`;
- BootInfo is validated (magic/version/size) and consumed via base+offset;
- a wide-store aligned memory clear succeeds;
- syscall/timer handlers still round-trip via `IRET`.

8 Section 5 v5: Kernel Stub & Early Boot (Protection + Dual Images)

Purpose

v5 adds the final foundational OS boundary:

kernel/user privilege separation and a dual-image boot model.

In v5:

- kernel boots privileged (K=1), owns vectors and BootInfo;
- user code runs unprivileged (K=0) in a user-safe region;
- user requests kernel services via `SYSCALL` (vector 0x00);
- illegal user access to vectors/MMIO triggers a protection trap to a dedicated vector.

8.1 Compatibility Contract

- **CPU baseline: v5** (K flag + protection traps exist).
- v3/v4 vector layout preserved: vectors `0x0000..0x00FF`, reset `0x0100`.
- Toolchain v5 supports **dual-image outputs**: `kernel.bin` + `user.bin`.
- Boot ABI from v4 remains: `kernel_main(bootinfo_ptr)` with `R0=bootinfo_ptr`.

8.2 User-mode memory access rules (normative)

- **Vectors:** `0x0000..0x00FF` (privileged-only)
- **User memory window:** `0x0100..0xFDFF` (user-accessible when `K=0`)
- **Reserved/MMIO:** `0xFE00..0xFFFF` (privileged-only)

When `K=0`, any fetch or data access into `0x0000..0x00FF` or `0xFE00..0xFFFF` triggers a protection trap to vector `0x03`.

Kernel stack convention (non-normative). The kernel *may* use high RAM (e.g., `0xF800..0xFDFF`) as its stack/critical scratch space, but this is a **software convention** rather than a hardware-enforced exclusion in v5.

8.3 Reset convention (v5 alignment with CPU roadmap)

On reset, the CPU state follows the v5 roadmap:

- `PC=0x0100`, `SP=FP=0xFDFF`, `EPC=0`
- `FLAGS.ZNVC=0`, `FLAGS.I=1`, `FLAGS.K=1`
- `CAUSE=0`, `BADADDR=0`

8.4 Recommended placement convention (non-normative)

To keep a clean software separation, the kernel *should* place its code/data in low memory (commonly below `0x8000`) and load the user image at a higher base (recommended `user_base=0x8000`). This is a placement convention, not a CPU-enforced boundary; the kernel must therefore avoid relying on secrecy for data stored in the user-accessible RAM window.

8.5 Required vectors (v5)

ID	Handler	Requirement
`0x00`	`syscall_handler`	Must end with `IRET`
`0x01`	`timer_handler`	Must end with `IRET`
`0x03`	`prot_fault_handler`	Deterministic termination and fault signature

8.6 BootInfo v5 extension (normative)

BootInfo keeps the same header and extends with user-image fields (version=5, larger size). Key additional fields (conceptually):

- `user_base` (must be within `0x0100..0xFDFF`; recommended `0x8000`),
- `user_size`,
- `user_entry`,
- `user_stack_top` (recommended 0xFDFF),
- `prot_vector` (0x03).

8.7 Kernel→user transition (normative)

After kernel initialization, kernel must:

1. validate BootInfo v5 user fields (`0x0100..0xFDFF` range + 8-byte alignment);
2. prepare user stack and entry;
3. set `K := 0`;
4. jump to `user_entry` in the RAM window.

8.8 Protection fault signature (normative)

On protection fault (vector 0x03), kernel must record deterministically:

Reg	Meaning
`R0`	Fault code (`0x50` protection violation)
`R1`	Vector ID (`0x03`)
`R2`	Location identifier (EPC snapshot if available, else constant)
`R3`	SP snapshot
`R4`	FP snapshot

8.9 Definition of Done (v5)

- dual-image boot reaches user entry in user mode (`K=0`);
- user syscall enters kernel via vector 0x00 and returns via `IRET`;
- user access to vectors/MMIO traps to vector 0x03 with correct fault signature.