IEN-002 - Stage 1 — Firmware Initialization (BIOS & UEFI)

Before Linux, before GRUB, and even before the operating system exists, the computer must first prepare itself to execute software.

This responsibility belongs to the system firmware.

For decades this firmware was known as the **Basic Input/Output System (BIOS)**. Modern systems have largely replaced BIOS with the **Unified Extensible Firmware Interface (UEFI)**, but regardless of the implementation, both serve the same fundamental purpose:

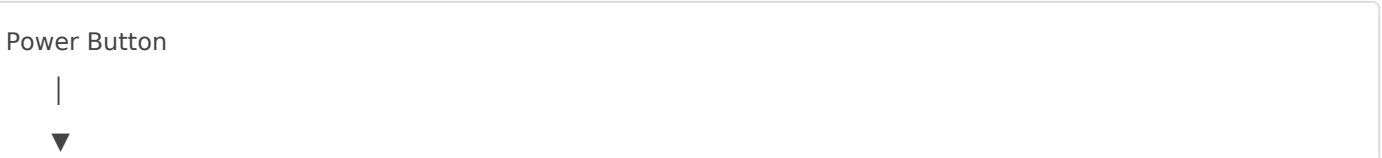
> Initialize the hardware and locate a bootable operating system.

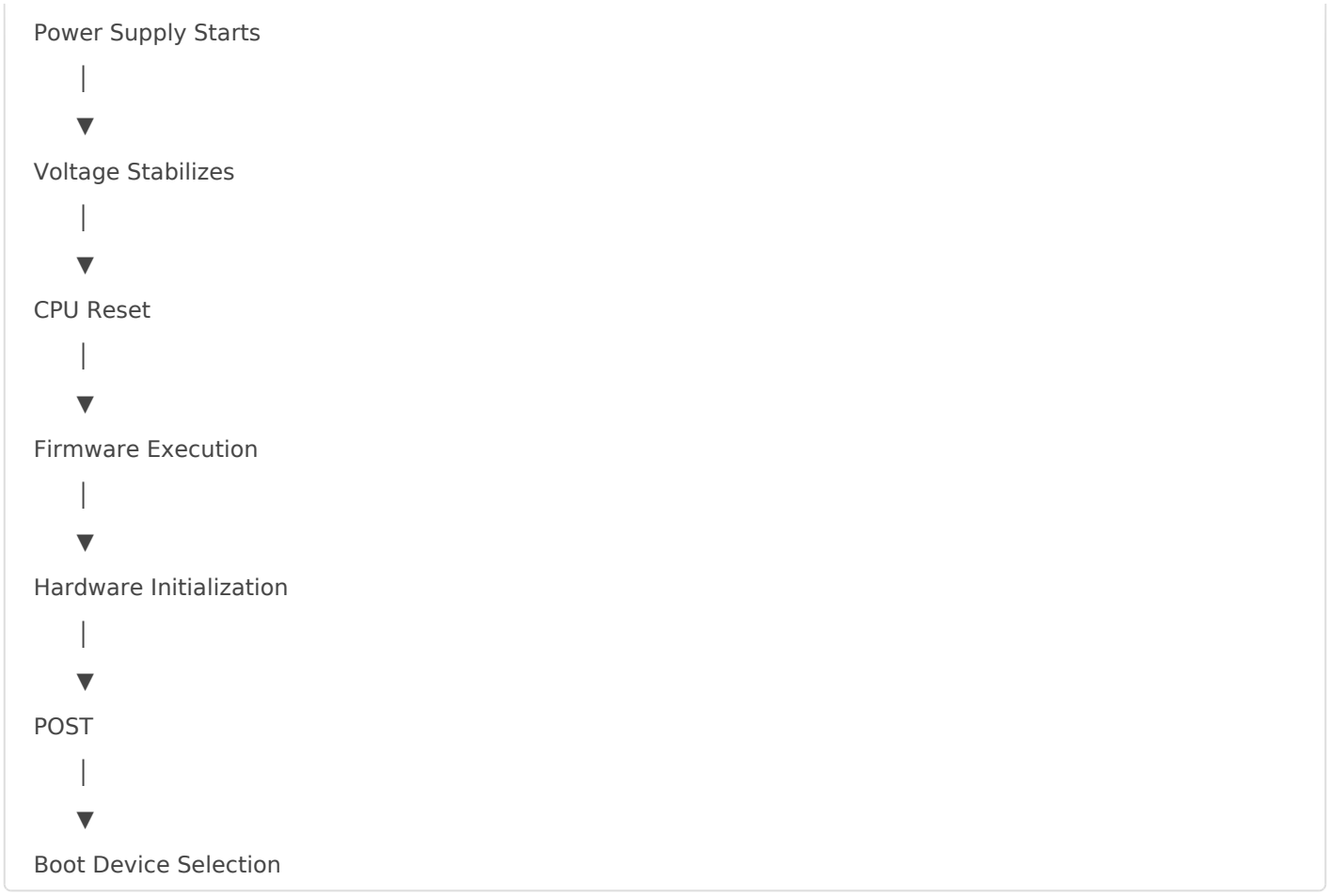
Without firmware, the processor has no knowledge of memory, storage devices, keyboards, displays, or even where the operating system is stored.

The firmware acts as the bridge between powered hardware and the operating system.

What Happens When You Press the Power Button?

Although pressing the power button appears to be a single action, it actually triggers a carefully coordinated sequence of hardware events.





Let's examine each step.

Step 1 — Power Supply Initialization

When the power button is pressed, the motherboard sends a signal to the power supply unit (PSU).

The PSU does not immediately provide full power to the system.

Instead, it gradually stabilizes all required voltage rails, such as:

- +12V
- +5V
- +3.3V

Only after these voltages become stable does the PSU assert a signal commonly called **Power Good (PWR_OK)**.

This signal tells the motherboard:

> "The electrical power is stable. The processor may begin execution."

If the PSU never asserts this signal, the CPU never starts.

Step 2 — CPU Reset

Once power is stable, the processor exits its reset state.

An important question arises:

How does the CPU know what instruction to execute first?

The answer is surprisingly simple.

The processor does **not** search the hard drive.

It does **not** search memory.

Instead, every CPU contains a predefined **Reset Vector**.

Immediately after reset, the CPU jumps to this fixed memory location.

Historically, on x86 processors, this address is:

```
FFFF:0000
```

or, in physical addressing,

```
0xFFFFFFF0
```

This location is mapped to firmware stored in non-volatile memory on the motherboard.

The CPU therefore begins executing firmware instructions—not Linux.

Engineering Insight

Many engineers imagine the CPU "looking for Linux."

It doesn't.

The CPU blindly executes the instruction located at the reset vector.

Everything else—including loading Linux—is performed later by firmware.

Step 3 — Firmware Execution

At this point the firmware begins running.

Depending on the system, this firmware may be:

- Legacy BIOS
- Modern UEFI firmware

Although their implementations differ significantly, both begin by initializing the computer's hardware.

Typical initialization includes:

- CPU configuration
- Memory controller initialization
- RAM detection
- PCIe device discovery
- USB controller initialization
- SATA/NVMe controller initialization
- Keyboard detection
- Display initialization

Without this initialization, the operating system would have no usable hardware.

POST (Power-On Self-Test)

One of the firmware's first major responsibilities is performing the **Power-On Self-Test**, commonly called **POST**.

POST verifies that essential hardware components are functional before attempting to boot an operating system.

Typical checks include:

Component	Purpose
CPU	Verify processor responds correctly
RAM	Detect installed memory
Video Adapter	Initialize display output
Keyboard	Verify input device
Storage Controllers	Detect disks
Firmware Integrity	Validate firmware operation

If a critical failure occurs, the boot process stops immediately.

Examples include:

- Missing RAM
- CPU initialization failure
- Corrupted firmware
- No graphics initialization (depending on platform)

Many servers report POST failures through:

- Beep codes
- Seven-segment diagnostic displays
- Front-panel LEDs
- IPMI/BMC event logs

Enterprise servers from vendors such as Dell, HPE, Lenovo, Inspur, and Supermicro often expose POST events through their management controllers.

Hardware Discovery

After POST completes successfully, firmware begins discovering available hardware.

Modern systems may contain dozens or even hundreds of devices.

Examples include:

CPU

Memory

PCIe Devices

NICs

RAID Controllers

HBAs

NVMe Drives

USB Controllers

Graphics Adapters

Each discovered device receives system resources such as:

- Memory addresses
- Interrupt assignments
- PCI bus numbers

Only after this discovery phase can the operating system communicate with the hardware.

BIOS vs UEFI

Both BIOS and UEFI prepare the system for booting, but they are fundamentally different architectures.

BIOS	UEFI
Introduced in the 1980s	Modern firmware architecture
16-bit environment	32-bit or 64-bit environment
MBR partition table	GPT partition table
Maximum disk size ≈ 2 TB	Supports very large disks
Limited firmware functionality	Extensible firmware services
Loads boot code from the MBR	Loads EFI executables from the EFI System Partition

Legacy BIOS executes the first sector of the selected boot disk, known as the **Master Boot Record (MBR)**.

UEFI works differently.

Instead of executing raw boot code from the first sector, it loads an EFI application stored inside a dedicated **EFI System Partition (ESP)**.

For Linux systems, this EFI application is often GRUB or another EFI-compatible bootloader.

Engineering Insight

One common misconception is that UEFI is simply a "new BIOS."

In reality, UEFI is a complete firmware platform with its own filesystem support, boot manager, runtime services, and application loader. Unlike BIOS, which merely executes the first sector of a disk, UEFI can directly read files from the EFI System Partition and launch EFI executables.

Selecting the Boot Device

Once hardware initialization is complete, the firmware must decide **which device to boot from**.

Typical boot devices include:

- SATA SSD
- NVMe SSD
- SAS Disk
- USB Flash Drive
- DVD
- PXE Network Boot
- iSCSI Boot
- Fibre Channel SAN

The firmware follows its configured boot order.

For example:

```
1. NVMe SSD
2. SATA SSD
3. PXE Network
4. USB
```

It attempts each device until a valid boot target is found.

If no bootable device is available, booting stops before Linux is ever loaded.

Typical messages include:

```
No Boot Device Found
```

or

```
Operating System Not Found
```

or

```
Insert Boot Media
```

These errors are frequently mistaken for Linux problems, but they actually indicate that the firmware could not locate a bootable operating system.

Where Can Things Go Wrong?

Understanding common failures at this stage helps narrow down troubleshooting quickly.

Symptom	Likely Cause	Linux Running?
System does not power on	PSU or motherboard issue	❌ No
Continuous beep codes	Hardware failure detected during POST	❌ No
Memory training failure	Defective or incompatible RAM	❌ No
"No Boot Device Found"	Disk not detected or incorrect boot order	❌ No
"Operating System Not Found"	Missing or damaged bootloader	❌ Not yet

Notice that **Linux has not started in any of these scenarios**. Rebuilding `initramfs`, reinstalling GRUB, or repairing filesystems would not resolve problems that occur before the firmware successfully hands control to the bootloader.

Summary

The firmware stage is responsible for preparing the system to run an operating system. It initializes hardware, performs POST, discovers available devices, and selects a bootable storage device. Only after these tasks are completed does it transfer control to the bootloader.

This stage forms the foundation of the entire boot process. If firmware initialization fails, Linux never has the opportunity to start.

Revision #2
Created 4 July 2026 04:10:16 by Kapak Maut
Updated 4 July 2026 04:12:28 by Kapak Maut