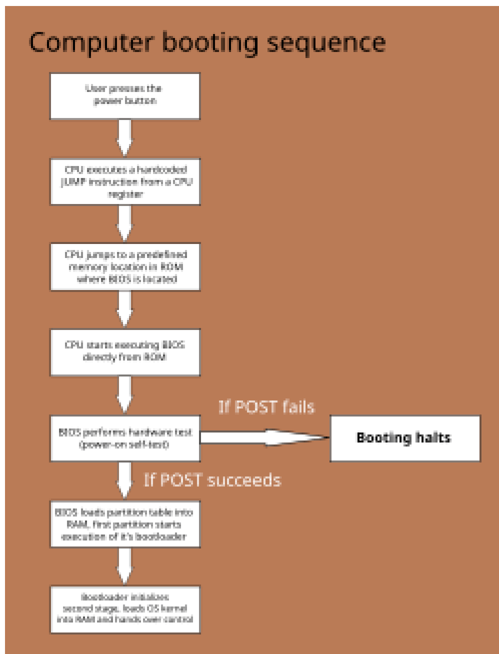
Booting

By Contributors to Wikimedia projects

Published: 2002-02-25 · Archived: 2026-04-05 22:48:38 UTC

This article is about bootstrapping operating systems. For the general concept, see Bootstrapping.



A flow diagram of a computer booting

In computing, **booting** is the process of starting a computer as initiated via hardware such as a physical button on the computer or by a software command, first described in the 1950s as the "bootstrap technique."[1] After it is switched on, a computer's central processing unit (CPU) has no software in its main memory, so some process must load software into memory before it can be executed. This may be done by hardware or firmware in the CPU, or by a separate processor in the computer system. On some systems, a power-on reset (POR) does not initiate booting, and the operator must initiate booting after POR completes. IBM uses the term **Initial Program Load** (**IPL**) on some[nb 1] product lines.

Restarting a computer is also called *rebooting*, which can be "hard", e.g., after electrical power to the CPU is switched from off to on, or "soft", where the power is not cut. On some systems, a soft boot may optionally clear RAM to zero. Both hard and soft booting can be initiated by hardware, such as a button press, or by a software command. Booting is complete when the operative runtime system, typically the operating system and some applications,[nb 2] is attained.

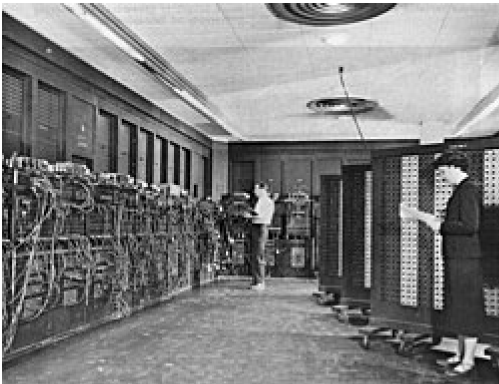
The process of returning a computer from a state of sleep (suspension) does not involve booting; however, restoring it from a state of hibernation does. Minimally, some embedded systems do not require a noticeable boot sequence to begin functioning, and when turned on, may simply run operational programs that are stored in read-

only memory (ROM). All computing systems are state machines, and a reboot may be the only method to return to a designated zero-state from an unintended, locked state.

In addition to loading an operating system or stand-alone utility, the boot process can also load a storage dump program for diagnosing problems in an operating system.

Boot is short for *bootstrap*[2][3] and derives from the phrase *to pull oneself up by one's bootstraps*.[4][5] The usage calls attention to the requirement that, if most software is loaded onto a computer by other software already running on the computer, some mechanism must exist to load the initial software onto the computer.[6] Early computers used a variety of ad-hoc methods to get a small program (the "bootstrap loader" or "bootstrap") into memory to solve this problem. The invention of ROM of various types solved this paradox by allowing computers to be shipped with a start-up program, stored in the boot ROM of the computer, that could not be erased. Growth in the capacity of ROM has allowed ever more elaborate start-up procedures to be implemented.

The earliest known recorded use of "boot" as the shortened form for "bootstrap" is 1975.[7]



Switches and cables used to program ENIAC (1946)

There are many different methods available to load a short initial program into a computer. These methods range from simple, physical input to removable media that can hold more complex programs.

Pre integrated-circuit-ROM examples

[edit]

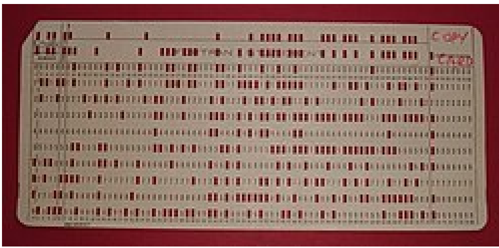
Early computers in the 1940s and 1950s were one-of-a-kind engineering efforts that could take weeks to program, and program loading was one of many problems that had to be solved. An early computer, ENIAC, had no program stored in memory but was set up for each problem by a configuration of interconnecting cables. Bootstrapping did not apply to ENIAC, whose hardware configuration was ready for solving problems as soon as power was applied.

The EDSAC system, the second stored-program computer to be built, used stepping switches to transfer a fixed program into memory when its start button was pressed. The program stored on this device, which David Wheeler completed in late 1948, loaded further instructions from punched tape and then executed them.[8][9]

First commercial computers

[edit]

The first programmable computers for commercial sale, such as the UNIVAC I and the IBM 701[10] included features to make their operation simpler. They typically included instructions that performed a complete input or output operation. The same hardware logic could be used to load the contents of a punch card (the most typical ones) or other input media, such as a magnetic drum or magnetic tape, that contained a bootstrap program by pressing a single button. This booting concept was called a variety of names for IBM computers of the 1950s and early 1960s, but IBM used the term "Initial Program Load" with the IBM 7030 Stretch[11] and later used it for their mainframe lines, starting with the System/360 in 1964.



Initial program load punched card for the IBM 1130 (1965)

The IBM 701 computer (1952–1956) had a "Load" button that initiated reading of the first 36-bit word into main memory from a punched card in a card reader, a magnetic tape in a tape drive, or a magnetic drum unit, depending on the position of the Load Selector switch. The left 18-bit half-word was then executed as an instruction, which usually read additional words into memory.[12][13] The loaded boot program was then executed, which, in turn, loaded a larger program from that medium into memory without further help from the human operator. The IBM 704,[14] IBM 7090,[15] and IBM 7094[16] had similar mechanisms, but with different load buttons for different devices. The term "boot" has been used in this sense since at least 1958.[17]

IBM System/3 console from the 1970s. Program load selector switch is lower left; Program load switch is lower right.

Other IBM computers of that era had similar features. For example, the IBM 1401 system (1959) used a card reader to load a program from a punched card. The 80 characters stored in the punched card were read into memory locations 001 to 080, then the computer would branch to memory location 001 to read its first stored instruction. This instruction was always the same: move the information in these first 80 memory locations to an assembly area where the information in punched cards 2, 3, 4, and so on, could be combined to form the stored program. Once this information was moved to the assembly area, the machine would branch to an instruction in location 080 (read a card) and the next card would be read and its information processed.

Another example was the IBM 650 (1953), a decimal machine, which had a group of ten 10-position switches on its operator panel that were addressable as a memory word (address 8000) and could be executed as an instruction. Thus, setting the switches to 7004000400 and pressing the appropriate button would read the first card in the card reader into memory (op code 70), starting at address 400 and then jump to 400 to begin executing the program on that card.[18] The IBM 7040 and 7044 have a similar mechanism, in which the Load button causes the instruction set up in the entry keys on the front panel is executed, and the channel that instruction sets up is given a command to transfer data to memory starting at address 00100; when that transfer finishes, the CPU jumps to address 00101.[19]

IBM's competitors also offered single-button program load.

- The CDC 6600 (c. 1964) had a *dead start* panel with 144 toggle switches; the dead start switch entered 12 12-bit words from the toggle switches to the memory of *peripheral processor* (*PP*) 0 and initiated the load sequence by causing PP 0 to execute the code loaded into memory.[20] PP 0 loaded the necessary code into its own memory and then initialized the other PPs.
- The GE 645 (c. 1965) had a "SYSTEM BOOTLOAD" button that, when pressed, caused one of the I/O controllers to load a 64-word program into memory from a diode read-only memory and deliver an interrupt to cause that program to start running.[21]
- The first model of the PDP-10 had a "READ IN" button that, when pressed, reset the processor and started an I/O operation on a device specified by switches on the control panel, reading in a 36-bit word giving a target address and count for subsequent word reads; when the read completed, the processor started executing the code read in by jumping to the last word read in.[22]

A noteworthy variation of this is found on the Burroughs B1700, where there is neither a boot ROM nor a hardwired IPL operation. Instead, after the system is reset, it reads and executes microinstructions sequentially from a cassette tape drive mounted on the front panel; this sets up a boot loader in RAM, which is then executed.[23] However, since this makes few assumptions about the system it can equally well be used to load diagnostic (Maintenance Test Routine) tapes which display an intelligible code on the front panel even in cases of gross CPU failure.[23]

IBM System/360 and successors

[edit]

In the IBM System/360 and its successors, including the current z/Architecture machines, the boot process is known as *Initial Program Load* (IPL).

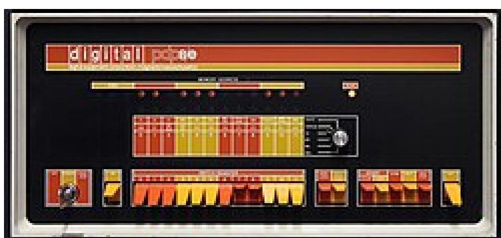
IBM coined this term for the 7030 (Stretch),[11] revived it for the design of the System/360, and continues to use it in those environments today.[24] In the System/360 processors, an IPL is initiated by the computer operator by selecting the three hexadecimal digit device address (CUU; C=I/O Channel address, UU=Control unit and Device address[nb 3]) followed by pressing the *LOAD* button. On the high end System/360 models, most[nb 4] System/370 and some later systems, the functions of the switches and the LOAD button are simulated using selectable areas on the screen of a graphics console, often[nb 5] an IBM 2250-like device or an IBM 3270-like device. For

example, on the System/370 Model 158, the keyboard sequence 0-7-X (zero, seven and X, in that order) results in an IPL from the device address that was keyed into the input area. The Amdahl 470V/6 and related CPUs supported four hexadecimal digits on those CPUs that had the optional second channel unit installed, for a total of 32 channels. Later, IBM would also support more than 16 channels.

The IPL function in the System/360 and its successors prior to IBM Z, and its compatibles, such as Amdahl's, reads 24 bytes from an operator-specified device into main storage starting at real address zero. The second and third groups of eight bytes are treated as Channel Command Words (CCWs) to continue loading the startup program (the first CCW is always simulated by the CPU and consists of a Read IPL command, 02h, with command chaining and suppress incorrect length indication being enforced). When the I/O channel commands are complete, the first group of eight bytes is then loaded into the processor's Program Status Word (PSW) and the startup program begins execution at the location designated by that PSW.[24] The IPL device is usually a disk drive, hence the special significance of the 02h read-type command, but exactly the same procedure is also used to IPL from other input-type devices, such as tape drives, or even card readers, in a device-independent manner, allowing, for example, the installation of an operating system on a brand-new computer from an OS initial distribution magnetic tape. For disk controllers, the 02h command also causes the selected device to seek to cylinder 0000h, head 0000h, simulating a Seek cylinder and head command, 07h, and to search for record 01h, simulating a Search ID Equal command, 31h; seeks and searches are not simulated by tape and card controllers, as for these device classes a Read IPL command is simply a sequential read command.

The disk, tape or card deck must contain a special program to load the actual operating system or standalone utility into main storage, and for this specific purpose, "IPL Text" is placed on the disk by the stand-alone DASDI (Direct Access Storage Device Initialization) program or an equivalent program running under an operating system, e.g., ICKDSF, but IPL-able tapes and card decks are usually distributed with this "IPL Text" already present.

IBM introduced some evolutionary changes in the IPL process, changing some details for System/370 Extended Architecture (S/370-XA) and later, and adding a new type of IPL for z/Architecture.



PDP-8/E front panel showing the switches used to load the bootstrap program

Minicomputers, starting with the Digital Equipment Corporation (DEC) PDP-5 and PDP-8 (1965) simplified design by using the CPU to assist input and output operations. This saved cost but made booting more complicated than pressing a single button. Minicomputers typically had some way to *toggle in* short programs by manipulating an array of switches on the front panel. Since the early minicomputers used magnetic-core memory, which did not lose its information when power was off, these bootstrap loaders would remain in place unless they were erased. Erasure sometimes happened accidentally when a program bug caused a loop that overwrote all of memory.

Other minicomputers with such simple form of booting include Hewlett-Packard's HP 2100 series (mid-1960s), the original Data General Nova (1969), and DEC's PDP-4 (1962) and PDP-11 (1970).

As the I/O operations needed to cause a read operation on a minicomputer I/O device were typically different for different device controllers, different bootstrap programs were needed for different devices.

DEC later added, in 1971, an optional diode matrix read-only memory for the PDP-11 that stored a bootstrap program of up to 32 words (64 bytes). It consisted of a printed circuit card, the M792, that plugged into the Unibus and held a 32 by 16 array of semiconductor diodes. With all 512 diodes in place, the memory contained all "one" bits; the card was programmed by cutting off each diode whose bit was to be "zero". DEC also sold versions of the card, the BM792-Yx series, pre-programmed for many standard input devices by simply omitting the unneeded diodes.[25][26]

Following the older approach, the earlier PDP-1 has a hardware loader, such that an operator needs only push the *load* switch to instruct the paper tape reader to load a program directly into core memory. The PDP-7,[27] PDP-9,[28] and PDP-15[29] successors to the PDP-4 have an added Read-In button to read a program in from paper tape and jump to it. The Data General Supernova used front panel switches to cause the computer to automatically load instructions into memory from a device specified by the front panel's data switches, and then jump to loaded code.[30]

Early minicomputer boot loader examples

[edit]

In a minicomputer with a paper tape reader, the first program to run in the boot process, the boot loader, would read into core memory either the second-stage boot loader (often called a *Binary Loader*) that could read paper tape with checksum or the operating system from an outside storage medium. Pseudocode for the boot loader might be as simple as the following eight instructions:

1. Set the P register to 9
2. Check paper tape reader ready
3. If not ready, jump to 2
4. Read a byte from paper tape reader to accumulator
5. Store accumulator to address in P register
6. If end of tape, jump to 9
7. Increment the P register
8. Jump to 2

A related example is based on a loader for a Nicolet Instrument Corporation minicomputer of the 1970s, using the paper tape reader-punch unit on a Teletype Model 33 ASR teleprinter. The bytes of its second-stage loader are read from paper tape in reverse order.

1. Set the P register to 106
2. Check paper tape reader ready
3. If not ready, jump to 2

4. Read a byte from paper tape reader to accumulator
5. Store accumulator to address in P register
6. Decrement the P register
7. Jump to 2

The length of the second stage loader is such that the final byte overwrites location 7. After the instruction in location 6 executes, location 7 starts the second stage loader executing. The second stage loader then waits for the much longer tape containing the operating system to be placed in the tape reader. The difference between the boot loader and second stage loader is the addition of checking code to trap paper tape read errors, a frequent occurrence with relatively low-cost, "part-time-duty" hardware, such as the Teletype Model 33 ASR. (Friden Flexowriters were far more reliable, but also comparatively costly.)

Booting the first microcomputers

[edit]

The earliest microcomputers, such as the Altair 8800 (released first in 1975) and an even earlier, similar machine (based on the Intel 8008 CPU) had no bootstrapping hardware as such.[31] When powered up, the CPU would see memory that would contain random data. The front panels of these machines carried toggle switches for entering addresses and data, one switch per bit of the computer memory word and address bus. Simple additions to the hardware permitted one memory location at a time to be loaded from those switches to store bootstrap code. Meanwhile, the CPU was kept from attempting to execute memory content. Once correctly loaded, the CPU was enabled to execute the bootstrapping code. This process, similar to that used for several earlier minicomputers, was tedious and had to be error-free.[32]

Integrated circuit read-only memory era

[edit]

An Intel 2708 EPROM "chip" on a circuit board

The introduction of integrated circuit read-only memory (ROM), with its many variants, including mask-programmed ROMs, programmable ROMs (PROM), erasable programmable ROMs (EPROM), and flash

memory, reduced the physical size and cost of ROM. This allowed firmware boot programs to be included as part of the computer.

The Data General Nova 1200 (1970) and Nova 800 (1971) had a program load switch that, in combination with options that provided two ROM chips, loaded a program into main memory from those ROM chips and jumped to it.[30] Digital Equipment Corporation introduced the integrated-circuit-ROM-based BM873 (1974),[33] M9301 (1977),[34] M9312 (1978),[35] REV11-A and REV11-C,[36] MRV11-C,[37] and MRV11-D[38] ROM memories, all usable as bootstrap ROMs. The PDP-11/34 (1976),[39] PDP-11/60 (1977),[40] PDP-11/24 (1979),[41] and most later models include boot ROM modules.

An Italian telephone switching computer, called "Gruppi Speciali", patented in 1975 by Alberto Ciaramella, a researcher at CSELT,[42] included an (external) ROM. Gruppi Speciali was, starting from 1975, a fully single-button machine booting into the operating system from a ROM memory composed of semiconductors, not ferrite cores. Although the ROM device was not natively embedded in the computer of Gruppi Speciali, due to the design of the machine, it also allowed the single-button ROM booting in machines not designed for that (therefore, this "bootstrap device" was architecture-independent), e.g., the PDP-11. Storing the state of the machine after the switch-off was also in place, which was another critical feature in the telephone switching contest.[43]

Some minicomputers and superminicomputers include a separate console processor that bootstraps the main processor. The PDP-11/44 had an Intel 8085 as a console processor;[44] the VAX-11/780, the first member of Digital's VAX line of 32-bit superminicomputers, had an LSI-11-based console processor,[45] and the VAX-11/730 had an 8085-based console processor.[46] These console processors could boot the main processor from various storage devices.

Some other superminicomputers, such as the VAX-11/750, implement console functions, including the first stage of booting, in CPU microcode.[47]

Microprocessors and microcomputers

[edit]

Typically, a microprocessor will, after a reset or power-on condition, perform a start-up process that usually takes the form of "begin execution of the code that is found starting at a specific address" or "look for a multibyte code at a specific address and jump to the indicated location to begin execution". A system built using that microprocessor will have the permanent ROM occupying these special locations so that the system always begins operating without operator assistance. For example, Intel x86 processors always start by running the instructions beginning at F000:FFF0,[48][49] while for the MOS 6502 processor, initialization begins by reading a two-byte vector address at $FFFD (MS byte) and $FFFC (LS byte) and jumping to that location to run the bootstrap code.[50]

Apple Computer's first computer, the Apple 1, introduced in 1976, featured PROM chips that eliminated the need for a front panel for the boot process (as was the case with the Altair 8800) in a commercial computer. According

to Apple's ad announcing it, "No More Switches, No More Lights ... the firmware in PROMS enables you to enter, display and debug programs (all in hex) from the keyboard."[51]

Due to the expense of read-only memory at the time, the Apple II booted its disk operating systems using a series of very small incremental steps, each passing control onward to the next phase of the gradually more complex boot process. (See Apple DOS: Boot loader). Because so little of the disk operating system relied on ROM, the hardware was also extremely flexible and supported a wide range of customized disk copy protection mechanisms. (See Software Cracking: History.)

Some operating systems, most notably pre-1995 Macintosh systems from Apple, are so closely interwoven with their hardware that it is impossible to natively boot an operating system other than the standard one. This is the opposite extreme of the scenario using switches mentioned above; it is highly inflexible but relatively error-proof and foolproof as long as all hardware is working normally. A common solution in such situations is to design a boot loader that works as a program belonging to the standard OS that hijacks the system and loads the alternative OS. This technique was used by Apple for its A/UX Unix implementation and copied by various freeware operating systems and BeOS Personal Edition 5.

Some machines, like the Atari ST microcomputer, were "instant-on", with the operating system executing from a ROM. Retrieval of the OS from secondary or tertiary store was thus eliminated as one of the characteristic operations for bootstrapping. To allow system customizations, accessories, and other support software to be loaded automatically, the Atari's floppy drive was read for additional components during the boot process. There was a timeout delay that provided time to manually insert a floppy as the system searched for the extra components. This could be avoided by inserting a blank disk. The Atari ST hardware was also designed so the cartridge slot could provide native program execution for gaming purposes as a holdover from Atari's legacy making electronic games; by inserting the Spectre GCR cartridge with the Macintosh system ROM in the game slot and turning the Atari on, it could "natively boot" the Macintosh operating system rather than Atari's own TOS.

The IBM Personal Computer included ROM-based firmware called the BIOS; one of the functions of that firmware was to perform a power-on self test when the machine was powered up, and then to read software from a boot device and execute it. Firmware compatible with the BIOS on the IBM Personal Computer is used in IBM PC compatible computers. The UEFI was developed by Intel, originally for Itanium-based machines, and later also used as an alternative to the BIOS in x86-based machines, including Apple Macs using Intel processors.

Unix workstations originally had vendor-specific ROM-based firmware. Sun Microsystems later developed OpenBoot, later known as Open Firmware, which incorporated a Forth interpreter, with much of the firmware being written in Forth. It was standardized by the IEEE as IEEE standard 1275-1994; firmware that implements that standard was used in PowerPC-based Macs and some other PowerPC-based machines, as well as Sun's own SPARC-based computers. The Advanced RISC Computing specification defined another firmware standard, which was implemented on some MIPS-based and Alpha-based machines and the SGI Visual Workstation x86-based workstations.

Modern boot loaders

[edit]

When a computer is turned off, its software—including operating systems, application code, and data—remains stored on non-volatile memory. When the computer is powered on, it typically does not have an operating system or its loader in random-access memory (RAM). The computer first executes a relatively small program stored in read-only memory (ROM, and later EEPROM, NOR flash) which support execute in place, to initialize CPU and motherboard, to initialize the memory (especially on x86 systems), to initialize and access the storage (usually a block-addressed device, e.g. hard disk drive, NAND flash, solid-state drive) from which the operating system programs and data can be loaded into RAM, and to initialize other I/O devices.

The small program that starts this sequence is known as a *bootstrap loader*, *bootstrap* or *boot loader*. Often, multiple-stage boot loaders are used, during which several programs of increasing complexity load one after the other in a process of chain loading.

Some earlier computer systems, upon receiving a boot signal from a human operator or a peripheral device, may load a very small number of fixed instructions into memory at a specific location, initialize at least one CPU, and then point the CPU to the instructions and start their execution. These instructions typically start an input operation from some peripheral device (which may be switch-selectable by the operator). Other systems may send hardware commands directly to peripheral devices or I/O controllers that cause an extremely simple input operation (such as "read sector zero of the system device into memory starting at location 1000") to be carried out, effectively loading a small number of boot loader instructions into memory; a completion signal from the I/O device may then be used to start execution of the instructions by the CPU.

Smaller computers often use less flexible but more automatic boot loader mechanisms to ensure that the computer starts quickly and with a predetermined software configuration. In many desktop computers, for example, the bootstrapping process begins with the CPU executing software contained in ROM (for example, the BIOS of an IBM PC) at a predefined address (some CPUs, including the Intel x86 series are designed to execute this software after reset without outside help). This software contains rudimentary functionality to search for devices eligible to participate in booting, and load a small program from a special section (most commonly the boot sector) of the most promising device, typically starting at a fixed entry point such as the start of the sector.

Boot loaders may face peculiar constraints, especially in size; for instance, on the IBM PC and compatibles, the boot code must fit in the Master Boot Record (MBR) and the Partition Boot Record (PBR), which in turn are limited to a single sector; on the IBM System/360, the size is limited by the IPL medium, e.g., card size, track size.

On systems with those constraints, the first program loaded into RAM may not be sufficiently large to load the operating system and, instead, must load another, larger program. The first program loaded into RAM is called a first-stage boot loader, and the program it loads is called a second-stage boot loader. On many embedded CPUs, the CPU built-in boot ROM, sometimes called the zero-stage boot loader,[52] can find and load first-stage boot loaders.

First-stage boot loaders

[edit]

Examples of first-stage (hardware initialization stage) boot loaders include BIOS, UEFI, coreboot, Libreboot and Das U-Boot. On the IBM PC, the boot loader in the Master Boot Record (MBR) and the Partition Boot Record (PBR) was coded to require at least 32 KB[53][54] (later expanded to 64 KB[55]) of system memory and only use instructions supported by the original 8088/8086 processors.

Second-stage boot loaders

[edit]

Second-stage (OS initialization stage) boot loaders, such as shim,[56] GNU GRUB, rEFInd, BOOTMGR, Syslinux, and NTLDR, are not themselves operating systems, but are able to load an operating system properly and transfer execution to it; the operating system subsequently initializes itself and may load extra device drivers. The second-stage boot loader does not need drivers for its own operation, but may instead use generic storage access methods provided by system firmware such as the BIOS, UEFI or Open Firmware, though typically with restricted hardware functionality and lower performance.[57]

Many boot loaders (like GNU GRUB, rEFInd, Windows's BOOTMGR, Syslinux, and Windows NT/2000/XP's NTLDR) can be configured to give the user multiple booting choices. These choices can include different operating systems (for dual or multi-booting from different partitions or drives), different versions of the same operating system (in case a new version has unexpected problems), different operating system loading options (e.g., booting into a rescue or safe mode), and some standalone programs that can function without an operating system, such as memory testers (e.g., memtest86+), a basic shell (as in GNU GRUB), or even games (see List of PC Booter games).[58] Some boot loaders can also load other boot loaders; for example, GRUB loads BOOTMGR instead of loading Windows directly. Usually, a default choice is preselected with a time delay during which a user can press a key to change the choice; after this delay, the default choice is automatically run so normal booting can occur without interaction.

The boot process can be considered complete when the computer is ready to interact with the user, or the operating system is capable of running system programs or application programs.

First and second stages boot loaders

[edit]

Some boot loaders, such as Das U-Boot and iBoot, include both first- and second-stage boot functions.

Embedded and multi-stage boot loaders

[edit]

Many embedded systems must boot immediately. For example, waiting a minute for a digital television or a GPS navigation device to start is generally unacceptable. Therefore, such devices have software systems in ROM or flash memory so the device can begin functioning immediately; little or no loading is necessary, because the loading can be precomputed and stored on the ROM when the device is made.[*citation needed*]

Large and complex systems may have boot procedures that proceed in multiple phases until finally the operating system and other programs are loaded and ready to execute. Because operating systems are designed as if they never start or stop, a boot loader might load the operating system, configure itself as a mere process within that system, and then irrevocably transfer control to the operating system. The boot loader then terminates normally as any other process would.

Most computers are also capable of booting over a computer network. In this scenario, the operating system is stored on the disk of a server, and certain parts of it are transferred to the client using a simple protocol such as the Trivial File Transfer Protocol (TFTP). After these parts have been transferred, the operating system takes over control of the booting process.

As with the second-stage boot loader, network booting begins by using generic network access methods provided by the network interface's boot ROM, which typically contains a Preboot Execution Environment (PXE) image. No drivers are required, but the system functionality is limited until the operating system kernel and drivers are transferred and started. As a result, once the ROM-based booting has completed it is entirely possible to network boot into an operating system that itself does not have the ability to use the network interface.

IBM-compatible personal computers (PC)

[edit]

Windows To Go bootable flash drive, a Live USB example

The boot device is the storage device from which the operating system is loaded. A modern PC's UEFI or BIOS firmware supports booting from various devices, typically a local solid-state drive or hard disk drive via the GPT or Master Boot Record (MBR) on such a drive or disk, an optical disc drive (using El Torito), a USB mass storage device (USB flash drive, memory card reader, USB hard disk drive, USB optical disc drive, USB solid-state drive, etc.), or a network interface card (using PXE). Older, less common BIOS-bootable devices include floppy disk drives, Zip drives, and LS-120 drives. IBM-compatible PCs are examples that use horizontal integrated hardware and UEFI/BIOS firmware.

Typically, the system firmware (UEFI or BIOS) will allow the user to configure a *boot order*. If the boot order is set to "first, the DVD drive; second, the hard disk drive", then the firmware will try to boot from the DVD drive, and if this fails (e.g., because there is no DVD in the drive), it will try to boot from the local hard disk drive.

For example, on a PC with Windows installed on the hard drive, the user could set the boot order to the one given above, and then insert a Linux Live CD in order to try out Linux without having to install an operating system onto the hard drive. This is an example of dual booting, in which the user chooses which operating system to start after the computer has performed its power-on self-test (POST). In this example of dual booting, the user chooses by inserting or removing the DVD from the computer, but it is more common to choose which operating system to

boot by selecting from a boot manager menu on the selected device, by using the computer keyboard to select from a BIOS or UEFI boot menu, or both; the boot menu is typically entered by pressing F8 or F12 keys during the POST; the BIOS setup is typically entered by pressing F2 or DEL keys during the POST.[59][60]

Several devices are available that enable the user to *quick-boot* into what is usually a variant of Linux for various simple tasks such as Internet access; examples are Splashtop and Latitude ON.[61][62][63]

A hex dump of FreeBSD's boot0 MBR

Award Software BIOS from 2000 during booting

Upon starting, an IBM-compatible personal computer's x86 CPU, executes in real mode, the instruction located at reset vector (the physical memory address FFFF0h on 16-bit x86 processors[64] and FFFFFFF0h on 32-bit and 64-bit x86 processors[65][66]), usually pointing to the firmware (UEFI or BIOS) entry point inside the ROM. This memory location typically contains a jump instruction that transfers execution to the location of the firmware (UEFI or BIOS) start-up program. This program runs a power-on self-test (POST) to check and initialize required devices such as main memory (DRAM), the PCI bus and the PCI devices (including running embedded option ROMs). One of the most involved steps is setting up DRAM over SPD, further complicated by the fact that, at this point, memory is very limited.

After initializing required hardware, the firmware (UEFI or BIOS) goes through a pre-configured list of non-volatile storage devices ("boot device sequence") until it finds one that is bootable.

Once the BIOS has found a bootable device it loads the boot sector to linear address 7C00h (usually segment:offset 0000h:7C00h,[53][55]:29 but some BIOSes erroneously use 07C0h:0000h[*citation needed*]) and transfers execution to the boot code. In the case of a hard disk, this is referred to as the Master Boot Record (MBR). The conventional MBR code checks the MBR's partition table for a partition set as *bootable*[nb 6] (the one

with *active* flag set). If an active partition is found, the MBR code loads the boot sector code from that partition, known as Volume Boot Record (VBR), and executes it. The MBR boot code is often operating-system specific.

A bootable MBR device is defined as one that can be read from, and where the last two bytes of the first sector contain the little-endian word AA55h,[nb 7] found as byte sequence 55h, AAh on disk (also known as the MBR boot signature), or where it is otherwise established that the code inside the sector is executable on x86 PCs.

The boot sector code is the first-stage boot loader. It is located on fixed disks and removable drives, and must fit into the first 446 bytes of the Master Boot Record in order to leave room for the default 64-byte partition table with four partition entries and the two-byte boot signature, which the BIOS requires for a proper boot loader — or even less, when additional features like more than four partition entries (up to 16 with 16 bytes each), a disk signature (6 bytes), a disk timestamp (6 bytes), an Advanced Active Partition (18 bytes) or special multi-boot loaders have to be supported as well in some environments. In floppy and superfloppy Volume Boot Records, up to 59 bytes are occupied for the Extended BIOS Parameter Block on FAT12 and FAT16 volumes since DOS 4.0, whereas the FAT32 EBPB introduced with DOS 7.1 requires even 87 bytes, leaving only 423 bytes for the boot loader when assuming a sector size of 512 bytes. Microsoft boot sectors therefore traditionally imposed certain restrictions on the boot process, for example, the boot file had to be located at a fixed position in the root directory of the file system and stored as consecutive sectors,[67][68] conditions taken care of by the `SYS` command and slightly relaxed in later versions of DOS.[68][nb 8] The boot loader was then able to load the first three sectors of the file into memory, which happened to contain another embedded boot loader able to load the remainder of the file into memory.[68] When Microsoft added LBA and FAT32 support, they even switched to a boot loader reaching over *two* physical sectors and using 386 instructions for size reasons. At the same time, other vendors managed to squeeze much more functionality into a single boot sector without relaxing the original constraints on only minimal available memory (32 KB) and processor support (8088/8086).[nb 9] For example, DR-DOS boot sectors are able to locate the boot file in the FAT12, FAT16 and FAT32 file system, and load it into memory as a whole via CHS or LBA, even if the file is not stored in a fixed location and in consecutive sectors.[69][53][70][71][72][nb 10][nb 9]

The VBR is often OS-specific; however, its main function is to load and execute the operating system boot loader file (such as `bootmgr` or `ntldr`), which is the second-stage boot loader, from an active partition. Then the boot loader loads the OS kernel from the storage device.

If there is no active partition, or the active partition's boot sector is invalid, the MBR may load a secondary boot loader which will select a partition (often via user input) and load its boot sector, which usually loads the corresponding operating system kernel. In some cases, the MBR may also attempt to load secondary boot loaders before trying to boot the active partition. If all else fails, it should issue an INT 18h[55][53] BIOS interrupt call (followed by an INT 19h just in case INT 18h would return) in order to give back control to the BIOS, which would then attempt to boot off other devices, attempt a remote boot via network.[53]

Many modern systems (Intel Macs and newer PCs) use UEFI.[73][74]

Unlike BIOS, UEFI (not **Legacy boot** via CSM) does not rely on boot sectors, UEFI system loads the boot loader (**EFI application** file in USB disk or in the EFI System Partition) directly,[75] and the OS kernel is loaded by the

boot loader.

SoCs, embedded systems, microcontrollers, and FPGAs

[edit]

An unlocked bootloader of an Android device, showing additional available options

Many modern CPUs, SoCs and microcontrollers (for example, TI OMAP) or sometimes even digital signal processors (DSPs) may have a boot ROM integrated directly into their silicon, so such a processor can perform a simple boot sequence on its own and load boot programs (firmware or software) from boot sources such as NAND flash or eMMC. It is difficult to hardwire all the required logic for handling such devices, so an integrated boot ROM is used instead in such scenarios. Also, a boot ROM may be able to load a boot loader or diagnostic program via serial interfaces like UART, SPI, USB and so on. This feature is often used for system recovery purposes, or it could also be used for initial non-volatile memory programming when there is no software available in the non-volatile memory yet. Many modern microcontrollers (e.g., flash memory controller on USB flash drives) have firmware ROM integrated directly into their silicon.

Some embedded system designs may also include an intermediary boot sequence step. For example, Das U-Boot may be split into two stages: the platform would load a small SPL (Secondary Program Loader), which is a stripped-down version of U-Boot, and the SPL would do some initial hardware configuration (e.g. DRAM initialization using CPU cache as RAM) and load the larger, fully featured version of U-Boot.[76] Such embedded systems may used highly customized and vertical integrated hardware and software, and their boot programs may be simpler.[77] Some CPUs and SoCs may not use CPU cache as RAM on boot process, they use an integrated boot processor to do some hardware configuration to reduce cost.[78]

It is also possible to take control of a system by using a hardware debug interface such as JTAG. Such an interface may be used to write the boot loader program into bootable non-volatile memory (e.g. flash) by instructing the processor core to perform the necessary actions to program non-volatile memory. Alternatively, the debug interface may be used to upload some diagnostic or boot code into RAM, and then to start the processor core and instruct it to execute the uploaded code. This allows, for example, the recovery of embedded systems where no software remains on any supported boot device, and where the processor does not have any integrated boot ROM. JTAG is a standard and popular interface; many CPUs, microcontrollers and other devices are manufactured with JTAG interfaces (as of 2009).[*citation needed*]

Some microcontrollers provide special hardware interfaces that cannot be used to take arbitrary control of a system or directly run code, but instead allow the insertion of boot code into bootable non-volatile memory (like flash memory) via simple protocols. Then, at the manufacturing phase, such interfaces are used to inject boot code (and possibly other code) into non-volatile memory. After system reset, the microcontroller begins to execute code programmed into its non-volatile memory, just like usual processors use ROMs for booting. Most notably, this technique is used by Atmel AVR microcontrollers, and by others as well. In many cases, such interfaces are implemented by hardwired logic. In other cases, such interfaces could be created by software running in integrated on-chip boot ROM from GPIO pins.

Most DSPs have a serial mode boot and a parallel mode boot, such as the host port interface (HPI boot).

In the case of DSPs, there is often a second microprocessor or microcontroller present in the system design, and this is responsible for overall system behavior, interrupt handling, dealing with external events, user interface, etc., while the DSP is dedicated to signal processing tasks only. In such systems, the DSP could be booted by another processor, which is sometimes referred as the *host processor* (giving name to a Host Port). Such a processor is also sometimes referred to as the *master*, since it usually boots first from its own memories and then controls overall system behavior, including booting of the DSP, and then further controls the DSP's behavior. The DSP often lacks its own boot memories and relies on the host processor to supply the required code instead. The most notable systems with such a design are cell phones, modems, audio and video players and so on, where a DSP and a CPU/microcontroller coexist.

Many FPGA chips load their configuration from an external configuration ROM, typically a serial EEPROM, on power-up.

Various measures have been implemented that enhance the security of the booting process. Some of them are made mandatory, others can be disabled or enabled by the end user. Traditionally, booting did not involve the use of cryptography. The security can be bypassed by unlocking the boot loader, which might or might not be approved by the manufacturer. Modern boot loaders make use of concurrency, meaning they can run multiple processor cores and threads at the same time, which adds extra layers of complexity to secure booting.

Matthew Garrett argued that booting security serves a legitimate goal, but in doing so chooses defaults that are hostile to users.[79]

- UEFI secure boot[80]
- Android Verified boot
- Samsung Knox

- Measured boot with the Trusted Platform Module, also known as "trusted boot".
- Intel BootGuard
- Disk encryption
- Firmware passwords

UART console of a TP-Link router with OpenWrt that is stuck in a bootloop

When debugging a concurrent and distributed system of systems, a **bootloop** (also written **boot loop** or **boot-loop**) is a diagnostic condition of an erroneous state that occurs on computing devices; when those devices repeatedly fail to complete the booting process and restart before a boot sequence is finished, a restart might prevent a user from accessing the regular interface.

> As the complexity of today's products increases, single projects, single departments or even single companies can no longer develop total products, causing concurrent and distributed development. Today and worldwide, industries are facing complex product development and its vast array of associated problems, relating to project organization, project control and product quality. Many processes will become distributed as well. The defect detection process, so important for measuring and eventually achieving product quality, is typically one of the first to experience problems caused by the distributed nature of the project. The distribution of defect detection activities over several parties introduces risks like the inadequate review of work products, occurrence of "blind spots" with respect to test coverage or over-testing of components. Lifecycle-wide coordination of defect detection is therefore needed to ensure effectiveness and efficiency of defect detection activities. —J.J.M. Trienekens; R.J. Kusters. (2004)[81]

Detection of an erroneous state

[edit]

The system might exhibit its erroneous state in, for example, an explicit bootloop or a blue screen of death, before recovery is indicated.[82] Detection of an erroneous state may require a distributed event store and stream-processing platform for real-time operation of a distributed system.

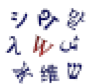
Recovery from an erroneous state

[edit]

An erroneous state can trigger bootloops; this state can be caused by misconfiguration from previously known-good operations. Recovery attempts from that erroneous state then enter a reboot, in an attempt to return to a known-good state. In Windows OS operations, for example, the recovery procedure was to reboot three times, the reboots needed to return to a usable menu.[83][84][82]

Recovery might be specified via Security Assertion Markup Language (SAML), which can also implement Single sign-on (SSO) for some applications; in the zero trust security model, identification, authorization, and authentication are separable concerns in an SSO session. When recovery of a site is indicated (viz., a blue screen of death is displayed on an airport terminal screen)[a] personal site visits might be required to remediate the situation.[81]

- Windows NT 4.0[85]
- Windows 2000[86]
- Windows Server[87]
- Windows 10[88]
- The Nexus 5X[89]
- Android 10: when setting a specific image as wallpaper, the luminance value exceeded the maximum of 255, which happened due to a rounding error during conversion from sRGB to RGB. This then crashed the SystemUI component on every boot.[90][91]
- Google Nest hub[92]
- LG smartphone bootloop issues
- On 19 July 2024, an update of CrowdStrike's Falcon software caused the 2024 CrowdStrike incident resulting in Microsoft Windows systems worldwide stuck in bootloops or recovery mode.[a]



Look up ***bootup*** in Wiktionary, the free dictionary.

- Bootstrapping § Computing
- Multi-booting
- Boot disk
- Bootkit
- Comparison of boot loaders
- Linux startup process
- Macintosh startup
- Microreboot
- Multi boot
- Network booting
- RedBoot
- Self-booting disk
- Windows startup process

1. ^ Jump up to: *a* *b* CrowdStrike reverted the content update at 05:27 UTC,[93] This left machines stuck in a boot loop or in recovery mode.[94] and devices booted after the revert were not affected.[84][95]

1. ^ E.g., System/360 through IBM Z, RS/6000 and System/38 through IBM Power Systems
2. ^ Including daemons.
3. ^ UU was often of the form Uu, U=Control unit address, u=Device address, but some control units attached only 8 devices; some attached more than 16. Indeed, the 3830 DASD controller offered 32-drive-addressing as an option.
4. ^ Excluding the 370/145 and 370/155, which used a 3210 or 3215 console typewriter.
5. ^ Only the S/360 used the 2250; the 360/85, 370/165 and 370/168 used a keyboard/display device compatible with nothing else.
6. ^ The active partition may contain a Second-stage boot loader, e.g., OS/2 Boot Manager, rather than an OS.
7. ^ The signature at offset `+1FEh` in boot sectors is `55h AAh`, that is `55h` at offset `+1FEh` and `AAh` at offset `+1FFh`. Since little-endian representation must be assumed in the context of IBM PC compatible machines, this can be written as 16-bit word `AA55h` in programs for x86 processors (note the swapped order), whereas it would have to be written as `55AAh` in programs for other CPU architectures using a big-endian representation. Since this has been mixed up numerous times in books and even in original Microsoft reference documents, this article uses the offset-based byte-wise on-disk representation to avoid any possible misinterpretation.
8. ^ The PC DOS 5.0 manual incorrectly states that the system files no longer need to be contiguous. However, for the boot process to work the system files still need to occupy the first two directory entries and the first three sectors of IBMBIO.COM still need to be stored contiguously. SYS continues to take care of these requirements.
9. ^ Jump up to: *a* *b* As an example, while the extended functionality of DR-DOS MBRs and boot sectors compared to their MS-DOS/PC DOS counterparts could still be achieved utilizing conventional code optimization techniques in assembly language up to 7.05, for the addition of LBA, FAT32 and LOADER support the 7.07 sectors had to resort to self-modifying code, opcode-level programming in machine language, controlled utilization of (documented) side effects, multi-level data/code overlapping and algorithmic folding techniques to squeeze everything into a single physical sector, as it was a requirement for backward- and cross-compatibility with other operating systems in multi-boot and chain load scenarios.
10. ^ There is one exception to the rule that DR-DOS VBRs will load the whole IBMBIO.COM file into memory: If the IBMBIO.COM file is larger than some 29 KB, trying to load the whole file into memory would result in the boot loader to overwrite the stack and relocated Disk Parameter Table (DPT/FDPB).[A] Therefore, a DR-DOS 7.07 VBR would only load the first 29 KB of the file into memory, relying on another loader embedded into the first part of IBMBIO.COM to check for this condition and load the remainder of the file into memory by itself if necessary. This does not cause compatibility problems, as IBMBIO.COM's size never exceeded this limit in previous versions without this loader.[A] Combined with a dual entry structure this also allows the system to be loaded by a PC DOS VBR, which would load only the first three sectors of the file into memory.

1. ^ *"boot-strap". Oxford English Dictionary. Vol. II (2nd ed.). Oxford: Clarendon Press. p. 407. “1953. A technique sometimes called the 'bootstrap technique'...”*

2. ^ *"bootstrap". Computer Dictionary of Information Technology. Archived from the original on 2019-08-05. Retrieved 2019-08-05.*
3. ^ *"Bootstrap". The Free Dictionary. Archived from the original on 2006-08-27. Retrieved 2008-08-27.*
4. ^ *"Pull oneself up by bootstraps". Idioms by The Free Dictionary. Archived from the original on 2018-10-05. Retrieved 2019-10-07.*
5. ^ *"Bootstrap Definition". Tech Terms. Archived from the original on 2020-05-10. Retrieved 2019-10-02.*
6. ^ *"Pull yourself up by your bootstraps". The Phrase Finder. Archived from the original on 2012-04-17. Retrieved 2010-07-15.*
7. ^ *"boot". Oxford English Dictionary. Vol. II (2nd ed.). Oxford: Clarendon Press. p. 405. "1975...The boot overlay code will overlay the first two instructions of the loader."*
8. ^ *Campbell-Kelly, Martin (1980). "Programming the EDSAC". IEEE Annals of the History of Computing. **2** (1): 7–36. doi:10.1109/mahc.1980.10009.*
9. ^ *Wilkes, Maurice V.; Wheeler, David J.; Gill, Stanley (1951). The Preparation of Programs for an Electronic Digital Computer. Addison-Wesley. Archived from the original on 2023-02-20. Retrieved 2020-09-25.*
10. ^ *Buchholz, Werner (1953). "The System Design of the IBM Type 701 Computer" (PDF). Proceedings of the I.R.E. **41** (10): 1273. Archived (PDF) from the original on 2022-10-09.*
11. ^ Jump up to: *a b* *"IBM 7619 Exchange". Reference Manual 7030 Data Processing System (PDF). IBM. August 1961. pp. 125–127. A22-6530-2. Archived (PDF) from the original on 2022-10-09.*
12. ^ *Principles of Operation Type 701 And Associated Equipment (PDF). IBM. 1953. p. 26. Archived (PDF) from the original on 2022-10-09. Retrieved 2012-11-09.*
13. ^ *Jeremy M. Norman (2005). From Gutenberg to the Internet. Norman. p. 436. ISBN 0-930405-87-0.*
14. ^ *704 Electronic Data-Processing Machine Manual of Operation (PDF). IBM. pp. 14–15. Archived (PDF) from the original on 2022-10-09.*
15. ^ *Operator's Guide for IBM 7090 Data Processing System (PDF). IBM. p. 34. Archived (PDF) from the original on 2022-10-09.*
16. ^ *IBM 7094 Principles of Operation (PDF). IBM. p. 146. Archived (PDF) from the original on 2022-10-09.*
17. ^ *Oxford English Dictionary. Oxford University. 1939.*
18. ^ *650 magnetic drum data-processing machine manual of operation (PDF). IBM. 1955. pp. 49, 53–54. Archived (PDF) from the original on 2022-10-09.*
19. ^ *Operator's Guide for IBM 7040-7044 Systems (PDF). IBM. p. 10. A22-6741-1. Archived (PDF) from the original on 2022-10-09.*
20. ^ *CONTROL DATA 6600 Computer System Reference Manual (PDF) (Second ed.). Control Data Corporation. August 1963. p. 53. Archived (PDF) from the original on 2022-10-09.*
21. ^ *GE-645 System Manual (PDF). General Electric. January 1968. Archived (PDF) from the original on 2022-10-09. Retrieved 2019-10-30.*
22. ^ *PDP-10 System Reference Manual, Part 1 (PDF). Digital Equipment Corporation. 1969. pp. 2–72. Archived (PDF) from the original on 2022-10-09. Retrieved 2012-11-09.*
23. ^ Jump up to: *a b* *Burroughs B 1700 Systems Reference Manual (PDF). Burroughs Corporation. November 1973. p. 1-14. Archived (PDF) from the original on 2022-10-09.*

24. ^ Jump up to: *a* *b* *z/Architecture Principles of Operation* (PDF). IBM. September 2005. pp. Chapter 17. Archived (PDF) from the original on 2022-10-09. Retrieved 2007-04-14.
25. ^ *BM792 read-only-memory and MR11~DB bootstrap loader* (PDF). Digital Equipment Corporation. January 1974. DEC-II-HBMAA-E-D. Archived (PDF) from the original on 2022-10-09.
26. ^ *PDP-11 Peripherals Handbook* (PDF). Digital Equipment Corporation. 1976. p. 4-25. Archived (PDF) from the original on 2022-10-09.
27. ^ *Programmed Data Processor-7 Users Handbook* (PDF). Digital Equipment Corporation. 1965. p. 143. Archived (PDF) from the original on 2022-10-09.
28. ^ *PDP-9 User Handbook* (PDF). Digital Equipment Corporation. January 1968. p. 10-3. Archived (PDF) from the original on 2022-10-09.
29. ^ *PDP-15 Systems Reference Manual* (PDF). Digital Equipment Corporation. August 1969. p. 10-3. Archived (PDF) from the original on 2022-10-09.
30. ^ Jump up to: *a* *b* *How To Use The Nova Computers* (PDF). Data General. April 1971. p. 2-30. Archived (PDF) from the original on 2022-10-09.
31. ^ *"Oldcomputers: Altair 8800b"*. Archived from the original on 2020-01-03. Retrieved 2019-12-10.
32. ^ Holmer, Glenn. *Altair 8800 loads 4K BASIC from paper tape*. Archived from the original on 2019-07-30. Retrieved 2016-05-02.
33. ^ *BM873 restart/loader* (PDF). Digital Equipment Corporation. April 1974. DEC-11-H873A-B-D. Archived (PDF) from the original on 2022-10-09.
34. ^ *M9301 bootstrap/terminator module maintenance and operator's manual* (PDF). Digital Equipment Corporation. June 1977. EK-M9301-TM-OO1. Archived (PDF) from the original on 2022-10-09.
35. ^ *M9312 bootstrap/terminator module technical manual* (PDF). Digital Equipment Corporation. March 1981. EK-M9312-TM-OO3. Archived (PDF) from the original on 2022-10-09.
36. ^ *Microcomputer Interfaces Handbook* (PDF). Digital Equipment Corporation. 1981. p. 17. Archived (PDF) from the original on 2022-10-09.
37. ^ "10 MRV11-C Read-Only Memory Module". *Microcomputer Products Handbook* (PDF). Digital Equipment Corporation. 1985. Archived (PDF) from the original on 2022-10-24. Retrieved 2022-06-12.
38. ^ "11 MRVll·D Universal Programmable Read.Only Memory". *Microcomputer Products Handbook* (PDF). Digital Equipment Corporation. 1985. Archived (PDF) from the original on 2022-10-24. Retrieved 2022-06-12.
39. ^ *PDP-11/34 system user's manual* (PDF). Digital Equipment Corporation. July 1977. pp. 1–5, 2-1 – 2-12. EK-11034-UG-001. Archived (PDF) from the original on 2022-10-09.
40. ^ *PDP-11/60 installation and operation manual* (PDF). Digital Equipment Corporation. February 1979. pp. 1–10, 2-29 – 2-34, 3-1 – 3-6. EK-11060-OP-003. Archived (PDF) from the original on 2022-10-09.
41. ^ *PDP-11/24 System Technical Manual* (PDF). Digital Equipment Corporation. June 1981. p. 1-6. EK-11024-TM-001. Archived (PDF) from the original on 2022-10-09.
42. ^ Ciaramella, Alberto. Device for automatically loading the central memory of electronic processors U.S. Patent No. 4,117,974. 1978-10-03. (submitted in 1975)
43. ^ *Alberto Ciaramella racconta il brevetto del boostrap dei computer concepito in CSELT* [Alberto Ciaramella discusses the patent for bootstrapping computers conceived at CSELT] (in Italian). Archived from the original on 2021-11-13.

44. ^ *PDP-11/44 System Technical Manual (PDF). Digital Equipment Corporation. February 1979. p. 6-57. EK-KD11Z-TM-001. Archived (PDF) from the original on 2022-10-09.*
45. ^ *VAX-11/780 Hardware User's Guide (PDF). Digital Equipment Corporation. February 1979. 2.3 BOOTSTRAPPING and 3.6.1 Boot Command (B). EK-11780-UG-001. Archived (PDF) from the original on 2022-10-09.*
46. ^ *VAX-11/730 Central Processing Unit Technical Description (PDF). Digital Equipment Corporation. May 1982. p. 1-9. EK-KA730-TD-001. Archived (PDF) from the original on 2022-10-09.*
47. ^ *VAX-11/750 Software Installation Guide (PDF). Digital Equipment Corporation. December 1982. pp. 1-2 – 1-4, B-1 – B-8, C-1 – C-2. AA-K410C-TE. Archived (PDF) from the original on 2022-10-09.*
48. ^ *Osborne, Adam; Kane, Gerry (1981). Osborne 16-Bit Microprocessor Handbook (PDF). OSBORNE/McGraw-Hill. pp. 5–27. ISBN 0-931988-43-8. Archived (PDF) from the original on 2022-10-09. Retrieved 2019-08-23.*
49. ^ *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3 (3A, 3B, 3C & 3D): System Programming Guide (PDF). Archived (PDF) from the original on 2022-10-09.*
50. ^ *Osborne, Adam; Kane, Gerry (1981). Osborne 4&8-Bit Microprocessor Handbook. Osborne/McGraw-Hill. pp. 10–20. ISBN 0-931988-42-X.*
51. ^ Apple Ad, Interface Age, October 1976
52. ^ *"An Introduction to RISC-V Boot flow" (PDF). Retrieved 2024-09-04.*
53. ^ Jump up to: *a b c d e* *Paul, Matthias R. (1997-10-02) [1997-09-29]. "Caldera OpenDOS 7.01/7.02 Update Alpha 3 IBMBIO.COM - README.TXT and BOOT.TXT - A short description of how OpenDOS is booted". Archived from the original on 2003-10-04. Retrieved 2009-03-29.* [1]
54. ^ *Sakamoto, Masahiko (2010-05-13). "Why BIOS loads MBR into 7C00h in x86?". Glamenv-Septzen.net. Archived from the original on 2017-08-24. Retrieved 2012-08-22.*
55. ^ Jump up to: *a b c* *Compaq Computer Corporation; Phoenix Technologies Ltd; Intel Corporation (1996-01-11). "BIOS Boot Specification 1.01" (PDF). Archived (PDF) from the original on 2022-10-09. Retrieved 2017-12-21.*
56. ^ *Red Hat Bootloader Team. "UEFI shim loader". GitHub. Retrieved 2023-10-28.*
57. ^ *"Chapter 6 - Troubleshooting Startup and Disk Problems". Windows NT Server Resource Kit. Microsoft. Archived from the original on 2007-05-15.*
58. ^ *"Tint". coreboot. Archived from the original on 2010-12-28. Retrieved 2010-11-20.*
59. ^ *"List of PC brands with their corresponding hot-keys". www.disk-image.com. Archived from the original on 2020-11-11. Retrieved 2020-09-26.*
60. ^ *"How to Enter the BIOS on Any PC: Access Keys by Manufacturer | Tom's Hardware". www.tomshardware.com. Archived from the original on 2023-02-20. Retrieved 2020-09-26.*
61. ^ *Brown, Eric (2008-10-02). "MontaVista Linux drives Dell's quick-boot feature". The LinuxDevices Archive. linuxdevices.com. Retrieved 2010-11-20.*
62. ^ *Larabel, Michael (2008-06-14). "SplashTop Linux On HP, Dell Notebooks?". Phoronix. Archived from the original on 2016-10-05. Retrieved 2010-11-20.*
63. ^ *"Voodoo Envy's Instant-On IOS (powered by Splashtop)". YouTube. 2008-07-16. Archived from the original on 2021-11-13. Retrieved 2010-11-20.*
64. ^ *"iAPX 286 Programmer's Reference Manual" (PDF). Intel. 1983. Section 5.3 SYSTEM INITIALIZATION, p. 5-7. Archived (PDF) from the original on 2022-10-09. Retrieved 2019-08-23. "Since*

the CS register contains F000 (thus specifying a code segment starting at physical address F0000) and the instruction pointer contains FFF0, the processor will execute its first instruction at physical address FFFF0H."

65. ^ *"80386 Programmer's Reference Manual" (PDF). Intel. 1986. Section 10.2.3 First Instructions, p. 10-3. Archived (PDF) from the original on 2022-10-09. Retrieved 2013-11-03. "After RESET, address lines A31–20 are automatically asserted for instruction fetches. This fact, together with the initial values of CS:IP, causes instruction execution to begin at physical address FFFFFFF0H."*
66. ^ *"Intel 64 and IA-32 Architectures Software Developer's Manual" (PDF). Intel Corporation. May 2012. Section 9.1.4 First Instruction Executed, p. 2611. Archived (PDF) from the original on 2022-10-09. Retrieved 2012-08-23. "The first instruction that is fetched and executed following a hardware reset is located at physical address FFFFFFF0h. This address is 16 bytes below the processor's uppermost physical address. The EPROM containing the software-initialization code must be located at this address."*
67. ^ *Zbikowski, Mark; Allen, Paul; Ballmer, Steve; Borman, Reuben; Borman, Rob; Butler, John; Carroll, Chuck; Chamberlain, Mark; Chell, David; Colee, Mike; Courtney, Mike; Dryfoos, Mike; Duncan, Rachel; Eckhardt, Kurt; Evans, Eric; Farmer, Rick; Gates, Bill; Geary, Michael; Griffin, Bob; Hogarth, Doug; Johnson, James W.; Kermaani, Kaamel; King, Adrian; Koch, Reed; Landowski, James; Larson, Chris; Lennon, Thomas; Lipkie, Dan; McDonald, Marc; McKinney, Bruce; Martin, Pascal; Mathers, Estelle; Matthews, Bob; Melin, David; Mergentime, Charles; Nevin, Randy; Newell, Dan; Newell, Tani; Norris, David; O'Leary, Mike; O'Rear, Bob; Olsson, Mike; Osterman, Larry; Ostling, Ridge; Pai, Sunil; Paterson, Tim; Perez, Gary; Peters, Chris; Petzold, Charles; Pollock, John; Reynolds, Aaron; Rubin, Darryl; Ryan, Ralph; Schulmeisters, Karl; Shah, Rajen; Shaw, Barry; Short, Anthony; Slivka, Ben; Smirl, Jon; Stillmaker, Betty; Stoddard, John; Tillman, Dennis; Whitten, Greg; Yount, Natalie; Zeck, Steve (1988). "Technical advisors". The MS-DOS Encyclopedia: versions 1.0 through 3.2. By Duncan, Ray; Bostwick, Steve; Burgoyne, Keith; Byers, Robert A.; Hogan, Thom; Kyle, Jim; Letwin, Gordon; Petzold, Charles; Rabinowitz, Chip; Tomlin, Jim; Wilton, Richard; Wolverton, Van; Wong, William; Woodcock, JoAnne (Completely reworked ed.). Redmond, Washington, USA: Microsoft Press. ISBN 1-55615-049-0. LCCN 87-21452. OCLC 16581341.* (xix+1570 pages; 26 cm) (NB. This edition was published in 1988 after extensive rework of the withdrawn 1986 first edition by a different team of authors: *"The MS-DOS Encyclopedia (1988)". PCjs Machines. Archived from the original on 2018-10-14.*)
68. ^ Jump up to: *a b c* *Chappell, Geoff (January 1994). "Chapter 2: The System Footprint". In Schulman, Andrew; Pedersen, Amorette (eds.). DOS Internals. The Andrew Schulman Programming Series (1st printing, 1st ed.). Addison Wesley Publishing Company. ISBN 978-0-201-60835-9.* (xxvi+738+iv pages, 3.5"-floppy [2][3]) Errata: [4][5][6]
69. ^ *Rosch, Winn L. (1991-02-12). "DR DOS 5.0 - The better operating system?". PC Magazine. Vol. 10, no. 3. p. 241–246, 257, 264, 266. Retrieved 2019-07-26. "[…] SYS has been improved under DR DOS 5.0 so you don't have to worry about leaving the first cluster free on a disk that you want to make bootable. The DR DOS system files can be located anywhere on the disk, so any disk with enough free space can be set to boot your system. […]"* `{{cite magazine}}` : CS1 maint: deprecated archival service (link) (NB. The source attributes this to the SYS utility while in fact this is a feature of the advanced bootstrap loader in the boot sector. SYS just plants this sector onto the disk.)
70. ^ *Paul, Matthias R. (2001-01-17). "FAT32 in DR-DOS". opendos@delorie. Archived from the original on 2017-10-06. Retrieved 2017-10-06. "[…] The DR-DOS boot sector […] searches for the IBMBIO.COM*

*(DRBIOS.SYS) file and then loads the *whole* file into memory before it passes control to it. […]”*

71. ^ Paul, Matthias R. (2002-02-20). "Can't copy". opendos@delorie. Archived from the original on 2017-10-06. Retrieved 2017-10-06. *“[…] The DR-DOS boot sector loads the whole IBMBIO.COM file into memory before it executes it. It does not care at all about the IBMDOS.COM file, which is loaded by IBMBIO.COM. […] The DR-DOS boot sector […] will find the […] kernel files as long as they are logically stored in the root directory. Their physical location on the disk, and if they are fragmented or not, is don't care for the DR-DOS boot sector. Hence, you can just copy the kernel files to the disk (even with a simply COPY), and as soon as the boot sector is a DR-DOS sector, it will find and load them. Of course, it is difficult to put all this into just 512 bytes, the size of a single sector, but this is a major convenience improvement if you have to set up a DR-DOS system, and it is also the key for the DR-DOS multi-OS LOADER utility to work. The MS-DOS kernel files must reside on specific locations, but the DR-DOS files can be anywhere, so you don't have to physically swap them around each time you boot the other OS. Also, it allows to upgrade a DR-DOS system simply by copying the kernel files over the old ones, no need for SYS, no difficult setup procedures as required for MS-DOS/PC DOS. You can even have multiple DR-DOS kernel files under different file names stored on the same drive, and LOADER will switch between them according to the file names listed in the BOOT.LST file. […]”*
72. ^ Paul, Matthias R. (2017-08-14) [2017-08-07]. "The continuing saga of Windows 3.1 in enhanced mode on OmniBook 300". MoHPC - the Museum of HP Calculators. Archived from the original on 2017-10-06. Retrieved 2017-10-06. *“[…] the DR-DOS FDISK does not only partition a disk, but can also format the freshly created volumes and initialize their boot sectors in one go, so there's no risk to accidentally mess up the wrong volume and no need for FORMAT /S or SYS. Afterwards, you could just copy over the remaining DR-DOS files, including the system files. It is important to know that, in contrast to MS-DOS/PC DOS, DR-DOS has "smart" boot sectors which will actually "mount" the file-system to search for and load the system files in the root directory instead of expecting them to be placed at a certain location. Physically, the system files can be located anywhere and also can be fragmented. […]”*
73. ^ "Intel Platform Innovation Framework for EFI". Intel. Archived from the original on 2011-08-21. Retrieved 2008-01-07.
74. ^ "OpenBIOS - coreboot". coreboot.org. Archived from the original on 2013-03-18. Retrieved 2013-03-20.
75. ^ "UEFI - OSDev Wiki". wiki.osdev.org. Archived from the original on 2020-11-12. Retrieved 2020-09-26.
76. ^ "Overview – The four bootloader stages". ti.com. Texas Instruments. 2013-12-05. Archived from the original on 2014-12-23. Retrieved 2015-01-25.
77. ^ Nnamdi Ajah. "BITE-SIZED BOOTING FOR ARM EMBEDDED SYSTEMS". Gist. Retrieved 2025-09-03.
78. ^ "The boot process rxos 1.0rc1 documentation". Retrieved 2015-10-25.
79. ^ "mjg59 | Boot Guard and PSB have user-hostile defaults". mjg59.dreamwidth.org. Retrieved 2022-11-30.
80. ^ "Microsoft blocks UEFI bootloaders enabling Windows Secure Boot bypass". BleepingComputer. Retrieved 2022-12-11.
81. ^ Jump up to: *a* *b* J.J.M. Trienekens; R.J. Kusters (19–21 September 2003). Workshop: defect detection in distributed software development. Eleventh Annual International Workshop on Software Technology and Engineering Practice. doi:10.1109/STEP.2003.40.
82. ^ Jump up to: *a* *b* Tim Warren (2024-07-23). "Inside the 78 minutes that took down millions of Windows machines". The Verge.

83. ^ Joe Tidy (2024-07-20). "CrowdStrike IT outage affected 8.5 million Windows devices, Microsoft says". BBC News.
84. ^ Jump up to: a b Piet Kerkhofs (2024-07-19). "CrowdStrike Falcon and Microsoft blue screen issue updates". Eye Security. Retrieved 2024-07-19.
85. ^ Ruley, John D.; David Methvin; Tom Henderson; Martin Heller (1997). Networking Windows NT 4.0: Workstation and Server. Wiley. p. 257. ISBN 978-0-471-17502-5 – via Google Books.
86. ^ Shultz, Gregory (February 2001). "Disabling automatic reboot prevents possible reboot loop". Windows Professional. **6** (2). Element K Journals: 9. ProQuest 191083238.
87. ^ "New Windows Server updates cause DC boot loops, break Hyper-V". BleepingComputer. Retrieved 2022-05-17.
88. ^ Paul Wagenseil (2021-01-21). "Windows 10 update sending PCs into endless boot cycle: What to do". Tom's Guide. Retrieved 2022-05-20.
89. ^ Hollister, Sean (2021-10-19). "Google has tried everything but building the best phone". The Verge. Retrieved 2022-05-17.
90. ^ "'It was unintentional,' says creator of 'cursed' Android wallpaper". The Week. Retrieved 2022-05-19.
91. ^ Hager, Ryne (2020-06-01). "Google thinks it has solved the mystery of the cursed bootlooping wallpaper". Android Police. Retrieved 2022-05-19.
92. ^ Peckham, James (2022-03-29). "Google Nest Hub gets a new UI that's so fresh it could bootloop your smart display". Android Police. Retrieved 2022-05-19.
93. ^ "Statement on Falcon Content Update for Windows Hosts". crowdstrike.com. Retrieved 2024-07-19.
94. ^ Baran, Guru (2024-07-19). "CrowdStrike Update Pushing Windows Machines Into a BSOD Loop". Cyber Security News. Retrieved 2024-07-19.
95. ^ "Botched security update breaks Windows worldwide, causing BSOD and crashes". Neowin. 2024-07-19. Retrieved 2024-07-19.

Source: https://en.wikipedia.org/wiki/Booting