Exhibit A

Booting

From Wikipedia, the free encyclopedia

In computing, **booting** (also known as **booting up**) is the initial set of operations that a computer system performs after electrical power to the CPU is switched on or when the computer is reset. The process begins when a computer is turned on for the first time, is re-energized after being turned off, when it is reset or when the operator invokes a LOAD[NB 1] function from the console, and ends when the computer is ready to perform its normal operations. On modern general purpose computers, this can take tens of seconds and typically involves performing a power-on self-test, locating and initializing peripheral devices, and then finding, loading and starting an operating system. Many computer systems also allow these operations to be initiated by a software command without cycling power, in what is known as a soft reboot, though some of the initial operations might be skipped on a soft reboot. A **boot loader** is a computer program that loads the main operating system or runtime environment for the computer after completion of the self-tests.

The computer term *boot* is short for *bootstrap*[1][2] or *bootstrap load* and derives from the phrase *to pull oneself up by one's bootstraps*.[3] 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.[4] Early computers used a variety of ad-hoc methods to get a small program into memory to solve this problem. The invention of read-only memory (ROM) of various types solved this paradox by allowing computers to be shipped with a start up program that could not be erased. Growth in the capacity of ROM has allowed ever more elaborate start up procedures to be implemented.

On general purpose computers, the boot process begins with the execution of an initial program stored in boot ROMs or read in another fashion. In some older computers, the initial program might have been the application to run, if no operating system was used, or the operating system. In other computers, the initial program is a boot loader that may then load into random-access memory (RAM), from nonvolatile secondary storage (such as a hard disk drive) or, in some older computers, from a medium such as punched cards, punched tape, or magnetic tape, the binary code of an operating system or runtime environment and then execute it. If the boot loader is limited in its size and capabilities, it may, instead, load a larger and more capable secondary boot loader, which would then load the operating system or runtime environment. 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 ROM.

Contents

History

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

Switches and cables used to program ENIAC (1946)

Pre integrated-circuit-ROM examples

Early computers

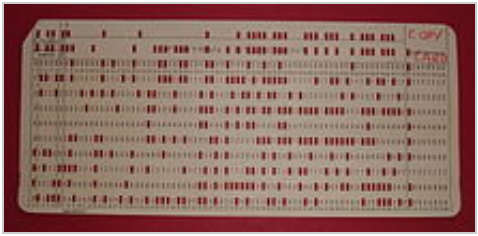
Early computers in the 1940s and 50s were one-of-a-kind engineering efforts that could take weeks to program and program loading was one of many problems that had to be solved. An early computer, ENIAC, as initially built, was not even programmable as we would think of such; its interconnections were made via cables to configure the hardware for the current problem. Bootstrapping in a stored-program computer simply did not apply. In 1960, the Ballistic Missile Early Warning System Display Information Processor (DIP) in Colorado Springs (before Cheyenne Mountain) ran only one program, which carried its own startup code. The program was stored as a bit image on a continuously running magnetic drum, and loaded in a fraction of a second. Core memory was probably cleared manually via the maintenance console, and startup from when power was fully up was very fast, only a few seconds. In its general design, the DIP compared roughly with a DEC PDP-8.

First commercial computers

The first programmable computers for commercial sale, such as the UNIVAC I and the IBM 701[5] 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 or other input media 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" starting with the System/360 in 1964.

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.[6][7] 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 term "boot" has been used in this sense since at least 1958.[8]



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

Other IBM computers of that era had similar features. For example, the IBM 1401 system (c. 1958) 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.

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

Another example was the IBM 650 (1953), a decimal machine, which had a group of ten 10-position switches on its operator panel which 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.[9]

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 words from the toggle switches to the memory of *peripheral processor* (*PP*) 0 and initiated the load sequence. 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.[10]
- 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.[11]

A noteworthy variation of this is found on the Burroughs B1700 where there is neither a bootstrap ROM nor a hardwired IPL operation. Instead, after the system is reset it reads and executes opcodes sequentially from a tape drive mounted on the front panel; this sets up a boot loader in RAM which is then executed. 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.

IBM System/360 and successors

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

This term was coined by IBM for the design of the System/360 (ca, 1965) and continues to be used in those environments today.[12] 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, U=Control unit address and u=Device address[NB 2]) followed by pressing the *LOAD* button. On 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 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 which was keyed into the input area. Amdahl 470V/6 and related CPUs supported four hexadecimal digits on those CPUs which 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, 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 implied). 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.[12] The IPL device is usually a disk drive, 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`, and to search for record 01h, thereby also simulating a stand-alone seek command, 07h, and a search ID equal command, 31h; seeks and searches are not simulated by tape and card controllers).

The disk, tape or card deck must contain a special program to load the actual operating system 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.

Minicomputers

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.

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

Other examples include early models of the Data General Nova (1969), PDP-11 (1970) and early microcomputers such as the Altair 8800 (1975). The Nova used 16 front panel switches and an enter pushbutton to manually load the first 22 addresses

into a core memory. DEC later added 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.[13][14]

Following the older approach, the earlier PDP-1 has a hardware loader, such that an operator need only push the "load" switch to instruct the paper tape reader to load a program directly into core memory.

Early minicomputer boot loader examples

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 a Teletype Model 33 ASR teleprinter as a paper tape reader. Note that the bytes of the 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

The earliest microcomputers, such as the Altair 8800 and an even earlier, similar machine (based on the Intel 8008 CPU) had no bootstrapping hardware as such. When started, the CPU would see memory that would contain executable code containing only binary zeros—memory was cleared by resetting when powering up. The front panels of these machines carried toggle switches, one switch per bit of the computer memory word. 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 was tedious and had to be error-free.

Integrated circuit read-only memory era

The boot process was revolutionized by 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. These allowed firmware boot programs to be shipped installed on the computer.

An Intel 2708 EPROM "chip" on a circuit board.

Apple Inc.'s first computer, the Apple 1 introduced in 1976, featured PROM chips that eliminated the need for a front panel for the boot process. 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."[15]

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 system.

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 Extensible Firmware Interface 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

When a modern computer is turned off, software, including operating systems, application code, and data, is stored on nonvolatile data storage devices such as hard drives, CDs, DVDs, flash memory cards (like SD cards), USB flash drives, and floppy disks. When the computer is powered on, it typically does not have an operating system in random access memory (RAM). The computer first executes a relatively small program stored in read-only memory (ROM) along with a small amount of needed data, to access the nonvolatile device or devices from which the operating system programs and data can be loaded into RAM.

The small program that starts this sequence is known as a *bootstrap loader*, *bootstrap* or *boot loader*. This small program's only job is to load other data and programs which are then executed from RAM. 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 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.

Boot loaders may face peculiar constraints, especially in size; for instance, on the IBM PC and compatibles, a boot sector should typically work in only 32 KB[16] (later relaxed to 64 KB[17]) of system memory and not use instructions not supported by the original 8088/8086 processors. The first stage of boot loaders located on fixed disks and removable drives 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 71 bytes, leaving only 441 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, conditions taken care of by the `SYS` command and slightly relaxed in later versions of DOS. 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. When they added LBA and FAT32 support, they even switched to a two-sector boot loader using 386 instructions. At the same time other vendors managed to squeeze much more functionality into a single boot sector without relaxing the original constraints on the only minimal available memory and processor support. 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.

Second-stage boot loader

Second-stage boot loaders, such as GNU GRUB, BOOTMGR, Syslinux, or 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.

Many boot loaders (like GNU GRUB, Windows's BOOTMGR, 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+) or even games (see List of PC Booter games).[18] 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. Typical modern personal computers boot in about one minute, of which about 15 seconds are taken by a power-on self-test (POST) and a preliminary boot loader, and the rest by loading the operating system and other software. Time spent after the operating system loading can be considerably shortened to as little as 3 seconds[19] by bringing the system up with all cores at once, as with coreboot.[20] Large servers may take several minutes to boot and start all their services.

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.

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.

Network booting

Main article: network booting

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. After these parts have been transferred, the operating system then takes over control of the booting process.

Boot devices (IBM PC)

See also: System partition and boot partition

Windows To Go bootable flash drive, a Live USB example

The boot device is the device from which the operating system is loaded. A modern PC BIOS supports booting from various devices, typically a local hard disk drive via the Master Boot Record (MBR) (and of several MS-DOS partitions on such a disk, or GPT through GRUB 2), an optical disc drive (using El Torito), a USB mass storage device (FTL-based flash drive, SD card, or multi-media card slot; hard disk drive, optical disc drive, etc.), or a network interface card (using PXE). Older, less common BIOS-bootable devices include floppy disk drives, SCSI devices, Zip drives, and LS-120 drives.

Typically, the 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 BIOS 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 drive.

For example, on a PC with Windows XP 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 — the user is choosing 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 CD from the computer, but it is more common to choose which operating system to boot by selecting from a BIOS or UEFI boot menu, by using the computer keyboard; the boot menu is typically entered by pressing ⌦ Delete or F11 keys during the POST.

Boot sequence of IBM PC compatibles

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[21] and `FFFFFFF0h` on 32-bit and 64-bit x86 processors[22][23]), usually pointing to the BIOS entry point inside the ROM. This memory location typically contains a jump instruction that transfers execution to the location of the BIOS start-up program. This program runs a power-on self-test (POST) to check and initialize required devices such as DRAM and the PCI bus (including running embedded ROMs). The most complicated step is setting up DRAM over SPI, made more difficult by the fact that at this point memory is very limited.

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

Coreboot splits the initialization and boot services into distinct parts, supporting "payloads" such as SeaBIOS, TianoCore, GRUB, and Linux directly (from flash).

Once the BIOS has found a bootable device it loads the boot sector to linear address 7C00h (usually segment:offset 0000h:7C00h, 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) and is by definition not operating-system specific. The conventional MBR code checks the MBR's partition table for a partition set as *bootable* (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.

A hex dump of FreeBSD's boot0 MBR

The VBR is often operating-system specific; however, in most operating systems its main function is to load and execute the operating system kernel, which continues startup.

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[17] 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 or invoke ROM BASIC.

Some systems (particularly newer Macintoshes and new editions of Microsoft Windows) use Intel's EFI. Also coreboot allows a computer to boot without having the firmware/BIOS constantly running in system management mode. 16-bit BIOS interfaces are required by certain x86 operating systems, such as DOS and Windows 3.1/95/98 (and all when not booted via UEFI). However, most boot loaders retain 16-bit BIOS call support.[24][25][26]

Other kinds of boot sequences

Award Software BIOS from 2000 during booting

Some other processors have other kinds of boot modes.

There are alternative techniques for booting CPUs and microcontrollers:

- Some modern CPUs and microcontrollers (for example, TI OMAP) or sometimes even DSPs may have boot ROM with boot code integrated directly into their silicon, so such a processor could perform quite a sophisticated boot sequence on its own and load boot programs from various sources like NAND flash, SD or MMC card and so on. It is hard to hardwire all the required logic for handling such devices, so an integrated boot ROM is used instead in such scenarios. Boot ROM usage

enables more flexible boot sequences than hardwired logic could provide. For example, the boot ROM could try to perform boot from multiple boot sources. Also, a boot ROM is often 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 when for some reasons usual boot software in non-volatile memory got erased. This technique could also be used for initial non-volatile memory programming when there is clean non-volatile memory installed and hence no software available in the system yet.

An unlocked Bootloader of the Android OS, showing additional options available for the user.

- 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).
- Some microcontrollers provide special hardware interfaces which can't be used to take arbitrary control of a system or directly run code, but instead they 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 are using 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 digital signal processors have the following boot modes:

- Serial mode boot
- Parallel mode boot, such as the host port interface (HPI boot)

In 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 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 controlling 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 are co-existing.

Many FPGA chips load their configuration from an external serial EEPROM ("configuration ROM") on power-up.

Quick boot

Several devices are available that enable the user to "quick-boot" to a usually Linux-powered OS for various simple tasks such as Internet access (such as Splashtop and Latitude ON).[27][28][29][30][31][32][33][34][35]

See also

- Boot disk
- Bootkit
- Comparison of boot loaders
- Das U-Boot
- El Torito (CD-ROM standard)
- Linux startup process
- Live CD
- Live USB
- Microreboot
- Multi boot
- Network booting
- PC booter
- Rebooting (computing)
- RedBoot
- Windows NT startup process
- Windows Vista startup process
- Windows To Go

Notes

1. ^ See IBM System/360 architecture#Operator controls
2. ^ Some control units attached only 8 devices; some attached more than 16. Indeed, the 3830 DASD controller offered 32-drive-addressing as an option.

References

1. ^ "Bootstrap" (http://dictionary.reference.com/search?r=2&q=bootstrap). Dictionary.com.
2. ^ "Bootstrap" (http://www.thefreedictionary.com/bootstrap). TheFreeDictionary.com.
3. ^ "pull oneself up by one's bootstraps - Wiktionary" (http://en.wiktionary.org/wiki/pull_oneself_up_by_one's_bootstraps). En.wiktionary.org. Retrieved 2013-05-25.
4. ^ "Phrase Finder" (http://www.phrases.org.uk/meanings/290800.html). phrases.org.uk.
5. ^ Buchholz, Werner (1953). "The System Design of the IBM Type 701 Computer" (http://bitsavers.org/pdf/ibm/701/Buchholz_IBM_701_System_Design_Oct53.pdf). *Proceedings of the I.R.E.* **41** (10): 1273.
6. ^ *Principles of Operation Type 701 And Associated Equipment* (http://bitsavers.org/pdf/ibm/701/24-6042-1_701_PrincOps.pdf). IBM. 1953. p. 26. Retrieved November 9, 2012.

7. ^ *From Gutenberg to the Internet*, Jeremy M. Norman, 2005, page 436, ISBN 0-930405-87-0
8. ^ *Oxford English Dictionary*. Oxford University.
9. ^ IBM 650 (http://bitsavers.trailing-edge.com/pdf/ibm/650/22-6060-2_650_OperMan.pdf)
10. ^ "GE-645 System Manual" (http://bitsavers.org/pdf/ge/GE-645/GE-645_SystemMan_Jan68.pdf). Retrieved November 6, 2012.
11. ^ *PDP-10 System Reference Manual, Part 1* (http://bitsavers.org/pdf/dec/pdp10/1970_PDP-10_Ref/1970PDP10Ref_Part1.pdf). Digital Equipment Corporation. 1969. pp. 2–72. Retrieved November 9, 2012.
12. ^ ***a*** ***b*** *z/Architecture Principles of Operation* (http://publibz.boulder.ibm.com/epubs/pdf/a2278324.pdf) (PDF). IBM. September 2005. Chapter 17. Retrieved 2007-04-14.
13. ^ PDP-11 Peripherals Handbook, DEC, 1975, p.4-25
14. ^ Photos of M792-YB Diode ROM bootstrapping card (http://decpicted.blogspot.com/2010/07/m792-yb-bootstrap-diode-matrix.html)
15. ^ Apple Ad, Interface Age, October 1976
16. ^ Masahiko Sakamoto (May 13, 2010). "Why BIOS loads MBR into 7C00h in x86?" (http://www.glamenv-septzen.net/en/view/6). Glamenv-Septzen.net. Retrieved 2012-08-22.
17. ^ ***a*** ***b*** Compaq Computer Corporation, Phoenix Technologies Ltd, Intel Corporation (1996-01-11). *BIOS Boot Specification 1.01* ([1] (https://www.acpica.org/download/specsbbs101.pdf)).
18. ^ "Tint" (http://www.coreboot.org/Tint). coreboot. Retrieved 20 November 2010.
19. ^ "FAQ - Why do we need coreboot?" (http://www.coreboot.org/FAQ#Why_do_we_need_coreboot_for_cluster_maintainance.3F). coreboot. Retrieved 20 November 2010.
20. ^ "Google tech talks - coreboot (aka LinuxBIOS): The Free/Open-Source x86 Firmware" (http://www.youtube.com/watch?v=X72LgcMpM9k). YouTube.
21. ^ "iAPX 286 Programmer's Reference Manual" (http://bitsavers.org/pdf/intel/80286/210498-001_1983_iAPX_286_Programmers_Reference_1983.pdf). Intel. 1983. Section 5.3 SYSTEM INITIALIZATION, p. 5-7. Retrieved November 3, 2013. "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."
22. ^ "80386 Programmer's Reference Manual" (http://bitsavers.org/pdf/intel/80386/230985-001_80386_Programmers_Reference_Manual_1986.pdf). Intel. 1986. Section 10.2.3 First Instructions, p. 10-3. Retrieved November 3, 2013. "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."
23. ^ "Intel® 64 and IA-32 Architectures Software Developer's Manual" (http://download.intel.com/products/processor/manual/325462.pdf). Intel Corporation. May 2012. Section 9.1.4 First Instruction Executed, p. 2611. Retrieved August 23, 2012. "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."
24. ^ "Intel Platform Innovation Framework for EFI" (http://www.intel.com/technology/framework/). Intel. Retrieved 2008-01-07.
25. ^ Intel Macintosh computers all have firmware with compatibility mode for legacy BIOS operations
26. ^ "OpenBIOS - coreboot" (http://www.coreboot.org/OpenBIOS). coreboot.org. Retrieved 2013-03-20.
27. ^ Brown, Eric (2008-10-02). "MontaVista Linux drives Dell's quick-boot feature" (http://archive.is/mKRQ). linuxdevices.com. Archived from the original (http://www.linuxdevices.com/news/NS2560585344.html) on 2012-09-07. Retrieved 20 November 2010.
28. ^ Larabel, Michael (June 14, 2008). "SplashTop Linux On HP, Dell Notebooks?" (http://www.phoronix.com/scan.php?page=article&item=splashtop_voodoo&num=1). Phoronix. Retrieved 20 November 2010.
29. ^ "Voodoo Envy's Instant-On IOS (powered by Splashtop)" (http://www.youtube.com/watch?v=InUpF5Uetfc). YouTube. Retrieved 20 November 2010.
30. ^ "Voodoo Envy 133 Laptop vs MacBook Air" (http://www.gadgets-reviews.com/voodoo-envy-133.html). gadgets-reviews.com. July 29, 2008. Retrieved 20 November 2010.

gadgets. Retrieved 20 November 2010.
31. ^ "Voodoopc homepage" (http://www.voodoopc.com/). Retrieved 20 November 2010.
32. ^ Brown, Eric (2008-10-03). "5-second Linux boots on low-powered hardware" (http://archive.is/cbsGm). Archived from the original (http://www.linuxdevices.com/news/NS7654890804.html) on 2013-01-28. Retrieved 20 November 2010.
33. ^ "Latitude ON" (http://www.youtube.com/watch?v=y40Z1mvGOt8). YouTube. Retrieved 20 November 2010.
34. ^ Brown, Eric (2008-11-07). "Linux boots in 2.97 seconds" (http://archive.is/IJRn). linuxdevices.com. Archived from the original (http://www.linuxdevices.com/news/NS5185504436.html) on 2012-09-14. Retrieved 20 November 2010.
35. ^ "News" (http://archive.is/WfQpd). linuxdevices.com. Archived from the original (http://www.linuxdevices.com/news/NS8282586707.html?kc=rss) on 2009-05-12. Retrieved 20 November 2010.

External links

- How Computers Boot Up (http://duartes.org/gustavo/blog/post/how-computers-boot-up)
- Practical boot loader tutorial for ATmega microcontrollers (http://www.societyofrobots.com/bootloader_50_robot.shtml)
- Booting with Grub (http://www.osdcom.info/content/view/33/39/) at OSDEV Community
- Tutorial on writing hello world boot loader (http://viralpatel.net/taj/tutorial/hello_world_bootloader.php)
- x86 BootStrap Programming Tutorial (http://www.vnutz.com/content/program_a_bootstrap_loader)
- Bootstrapping FreeBSD (http://www.khmere.com/freebsd_book/html/ch02.html)
- The Linux boot process unveiled (http://lateral.netmanagers.com.ar/stories/23.html)
- Mac OS X Boot Process (http://www.kernelthread.com/mac/osx/arch_boot.html)
- Jonathan de Boyne Pollard (2006). "The EFI boot process" (http://homepage.ntlworld.com./jonathan.deboynepollard/FGA/efi-boot-process.html). *Frequently Given Answers*.
- Jonathan de Boyne Pollard (2006). "The ARC boot process" (http://homepage.ntlworld.com./jonathan.deboynepollard/FGA/arc-boot-process.html). *Frequently Given Answers*.
- Jonathan de Boyne Pollard (1996). "The DOS and DOS/Windows boot processes" (http://homepage.ntlworld.com./jonathan.deboynepollard/FGA/dos-windows-boot-process.html). *Frequently Given Answers*.
- Jonathan de Boyne Pollard (2006). "The Windows NT 6 boot process" (http://homepage.ntlworld.com./jonathan.deboynepollard/FGA/windows-nt-6-boot-process.html). *Frequently Given Answers*.
- Windows Mobile 5.0 Soft Reset (http://www.pocketpcfaq.com/faqs/5.0/reset.htm)
- Pocket PC devices hard reset and soft reset (http://www.hardreset.eu/index_en.html)
- Cell phone, Tablet and Pocket PC devices hard reset and soft reset (http://www.hard-reset.com/)
- Understanding Multibooting (http://www.goodells.net/multiboot/)
- Code of a simple boot loader for students (http://code.google.com/p/akernelloader/)

Categories: Booting | BIOS | Boot loaders
