ROM and booting

As we know from the previous lectures, the role of RAM includes storing the code of the currently running program. However, at the time when a computer is powered on, RAM does not contain any useful data. The operating system, which also participates in memory management, isn't yet active either. To make setting up all of this possible, the computer motherboard contains an additional memory die. Clearly, it must be **non-volatile memory**, i.e. one which does not lose data during power-off periods.

That memory is usually of the **flash EEPROM** type. The name *EEPROM* can be a bit confusing, as it stands for: *electrically erasable programmable read-only memory.* The *read-only-memory* (*ROM*) part is here for historical reasons. Indeed, several production technologies ago, an important spot in computer design was occupied by memory in which data could be stored only once, at manufacturing time. However, later enhancements enabled that memory to be edited ("programmed") long after its creation (leading to PROM), then to be erased with UV light and programmed again (EPROM), up to the modern solutions in which such memory can be erased and programmed again without even taking it out of the computer.

The main content of the built-in non-volatile memory is *firmware* purposed to be executed as the first code after the computer is powered on. This plays the role of a "restricted operating system", enabling the processor to communicate with input/output devices on the motherboard (e.g. keyboard, disk, graphics card) until the true (much more complex) operating system is started and takes these tasks on.

For a long time, that initial firmware was usually based on a program for the IBM PC computer called BIOS — and, despite later extensions and adjustments, used to be still generally described as **BIOS** (*Basic Input/Output System*). In fact, that name can be still found in use, e.g. the abovementioned ROM memory is still sometimes called the *BIOS die.* However, in the last years, due to technical limitations typical to BIOS, its role has been taken over by a new firmware standard called **UEFI** (*Unified Extensible Firmware Interface*), developed and promoted in cooperation between multiple leading companies (including Intel, AMD, Lenovo, Dell, HP, Microsoft, and Apple).

Before going into the differences between BIOS and UEFI, we will describe the basic principles of the startup procedure of a computer and its operating system, which look similarly in both standards.

Booting

One of the main tasks of BIOS/UEFI is executing a **boot loader**, responsible for loading the operating system code into RAM, and then executing that code. In modern computers, this procedure often consists of **multiple stages**:

- A *first-stage* boot loader (BIOS/UEFI), instead of loading the operating system, loads another *second-stage* boot loader, which can be e.g.:
  - a boot loader for MS Windows (bootmgr), offering the user a choice between Windows versions (if there is more than one installed), as well as an option to run Windows in so-called *safe mode* (a restricted, and hence more stable, version of the system);
  - the GRUB boot loader, offering the user a choice between various UNIX-like operating systems (including Linux) and MS Windows — of course, if there is more than one such system installed.

- Only then, the proper operating system is invoked.

The second-stage boot loader is initially placed in the secondary memory, in a well-specified location of the particular storage device (e.g. for a hard drive, it is its first sector). (A part of a first-stage boot loader's work is to check all the computer storage drives for one that has such a boot sector). One can also boot from a pendrive, a CD/DVD disc, or even from another computer in the local network. (Earlier in the past, there were also other technologies in use, like floppy discs, or even punched tapes or cards). Notably, during an installation procedure for an operating system, the second-stage boot loader role is taken by the installer.

The POST self-check

Before BIOS/UEFI can proceed to the boot loader, it executes the **POST** procedure, which is the basic check of the crucial computer components. In particular, the checks involve the processor and main memory. In case any erorrs have been found, the appropriate message will be displayed on the screen — however, in case this could be impossible (e.g. due to a screen failure), the problems are additionally communicated with sound beeps. The exact meaning of these signals depends on the BIOS/UEFI manufacturer. Below, we list the meanings of sound signals according to one of the popular conventions, AMI BIOS:

Signal	**Error type**	**Comment**
1 long	no error	the POST test has finished with success
1 short	DRAM refresh error	e.g. an error while handling an interrupt
2 short	memory parity error	unexpected value of the checksum bit
3 short	RAM error	within the initial 64 kiB
4 short	system clock error	
5 short	CPU error	
6 short	A20 gate error	the *A20 gate* controls CPU access to memory addresses above 16 MiB; an error in it prevents CPU from switching between modes of operation
7 short	CPU virtual mode error	
8 short	video memory error	
9 short	ROM checksum error	ROM content has been corrupted
10 short	CMOS memory error	*CMOS* is a small memory for BIOS settings (which is non-volatile thanks to permanent power supply from the *CMOS battery*)
11 short	L2 cache error	
1 long, 2 short	video subsystem error	
1 long, 3 short	RAM error	outside the initial 64 kiB
1 long, 8 short	display error	

alternating high/low	insufficient power	insufficient CPU voltage or fan rotation speed

Occasionally, it's worth to check certain components even more thoroughly, which can be achieved with special diagnostic programs. Memory or hard drive can be tested from the operating system level; however, some tools (often these more thorough, e.g. `memtest86` for RAM) need to be launched from the BIOS/UEFI level.

Settings

After POST finishes, the user can enter the settings mode. To land there, one needs to press a proper key — usually `Esc` or one of the function keys (`F8`, `F12` etc.). The proper key depends on the BIOS/UEFI version; an information regarding which one is proper is displayed for some time on the startup screen.

In the settings mode, the user can adjust e.g. the clock frequency for the CPU, RAM, or system buses, the voltage powering the motherboard slots, or the computer fan rotation speed. Changing these parameters may improve computer efficiency — however, one should keep in mind that this brings a risk of unstable operation, or even damaging the hardware. Fortunately, at least as long as no component has been damaged, the settings mode allows restoring the factory defaults.

In that mode, the user can also set the order in which storage drives are checked for the boot loader. This is practically necessary if the top priority is currently assigned to the hard drive, while we want to boot from a pendrive (theoretically, an alternative is to take the hard drive out of the computer, though in practice it's clearly easier to change the settings). On the other hand, putting the hard drive on top of that list can save a few seconds of waiting during every normal computer startup. Similarly, some time can be saved by switching unnecessary devices — for example, when there is a non-connected slot on the motherboard. Another benefit from such settings change can be increasing security (e.g. by switching off all USB data transmission).

I/O interface

The last functionality of BIOS/UEFI to be mentioned here is defininig interrupt handlers — that is, defining a way for the CPU to communicate with the input/output devices. Until the 1990's, BIOS served as an intermediate layer between the operating system and I/O devices (which is reflected in its name). Currently, managing I/O devices is done inside the operating system; thanks to this, the user can attach a new device (and, if needed, install its software driver) without upgrading their BIOS/UEFI.

However, I/O management in BIOS/UEFI is still retained — e.g. to ensure backward compatibility (so that older programs can still execute), though most importantly to enable the abovementioned booting procedure during which BIOS/UEFI must load data from secondary storage without any help from the operating system (which is not yet running at that time).

BIOS vs. UEFI

From the user's viewpoint, the most noticeable difference between the newer UEFI and the older BIOS is appearance: while BIOS displays everything in ASCII mode, with no mouse support, UEFI offers a graphical interface (and, generally, more convenient usage). UEFI also removes some problems and limitations present in BIOS regarding handling hard drives (e.g. total size limited to 2 TiB, the number of so-called *primary partitions* limited to 4). However, the impact extends to other computer components. Also, some settings (e.g. Secure Boot) can on one hand improve security by making malware attacks significantly harder, but on the other hand may complicate or even prevent installing some operating systems.

UEFI also offers an operating mode compatible with the BIOS standard (*Compatibility Support Mode* — CSM; earlier also *Legacy BIOS*). Enabling it is necessary in case of some older components incompatible with the UEFI standard. On the other hand, some modules (e.g. Intel chipsets) do not offer compatibility with the BIOS standard. Similarly, the MS Windows operating system — starting from version 11 — requires the Secure Boot setting to support some of its functionalities, and that is only available in the UEFI mode, not in the BIOS compatibility modes.

Upgrades

As already mentioned, BIOS/UEFI on a particular computer can be upgraded (which may be necessary e.g. after replacing a hardware component). Saving a new version can be done in various ways (listed below starting from the most high-level ones):

1. From the level of an already running operating system (provided that the OS supports this);
2. From a special file placed on a pendrive, during booting, using the UEFI user interface;
3. From a special file placed on a pendrive, by pressing an appropriate button on the computer case (provided that the motherboard and the computer case both support this);
4. Directly to the EEPROM die (after opening the computer case), by using a dedicated device.

The more high-level ways are generally safer (e.g. the operating system and UEFI make an initial check of the update file, so that we do not risk e.g. using a "neighbouring" file of a wrong format by plain mistake). Still, such an update is potentially dangerous — it could make some I/O device inaccessible. If we're unlucky enough to pick a UEFI version incompatible with our motherboard, we may even lose way to turn the computer on; in such case, the only remaining ways to fix this are methods 3 and 4. The takeaway is that upgrading BIOS/UEFI without a direct reason is not recommended.