Memory Map (x86)

From OSDev Wiki

This article describes the contents of the computer's physical memory at the moment that the BIOS jumps to your bootloader code.

Contents

"Low" memory (< 1 MiB)

When a typical x86 PC boots it will be in Real Mode, with an active BIOS. During the time the CPU remains in Real Mode, IRQ0 (the clock) will fire repeatedly, and the hardware that is used to boot the PC (floppy, hard disk, CD, Network card, USB) will also generate IRQs. This means that during the PC boot process, the Real Mode IVT (see below) must be carefully preserved, because it is being used.

When the IVT is activated by an IRQ, it will call a BIOS routine to handle the IRQ. Bootloaders will also access BIOS functions. This means that the two memory workspaces that the BIOS uses (the BDA and the EBDA) must also be carefully preserved during boot. Also, every time the BIOS handles an IRQ0 (18 times a second), several bytes in the BDA get overwritten by the BIOS -- so do not attempt to store anything there while IRQs are active in Real Mode.

After all the BIOS functions have been called, and your kernel is loaded into memory somewhere, the bootloader or kernel may exit Real Mode forever (often by going into 32bit Protected Mode). If the kernel never uses Real Mode again, then the first 0x500 bytes of memory in the PC may be reused and overwritten. (However, it is very common to temporarily return to Real Mode in order to change the Video Display Mode.)

When the CPU is in Protected Mode, System Management Mode (SMM) is still invisibly active, and cannot be shut off. SMM also seems to use the EBDA. So the EBDA memory area should **never** be overwritten.

Note: the EBDA is a variable-sized memory area (on different BIOSes). If it exists, it is always immediately below 0xA0000 in memory. It is absolutely guaranteed to be less than 128 KiB in size. It is often 1 KiB. The biggest ones ever actually seen are 8 KiB. You can determine the size of the EBDA by using BIOS function INT 12h, or (often) by examining the word at 0x40E in the BDA (see below). Both of those methods will tell you the location of the bottom of the EBDA.

It should also be noted that your bootloader code is probably loaded and running in memory at physical addresses 0x7C00 through 0x7DFF. So that memory area is likely to also be unusable until execution has been transferred to a second stage bootloader, or to your kernel.

Overview

start	end	size	type	description
Low Memory (the first MiB)				
0x00000000	0x000003FF	1 KiB	RAM - partially unusable (see above)	Real Mode IVT (Interrupt Vector Table)
0x00000400	0x000004FF	256 bytes	RAM - partially unusable (see above)	BDA (BIOS data area)
0x00000500	0x00007BFF	almost 30 KiB	RAM (guaranteed free for use)	Conventional memory
0x00007C00 (typical location)	0x00007DFF	512 bytes	RAM - partially unusable (see above)	Your OS BootSector
0x00007E00	0x0007FFFF	480.5 KiB	RAM (guaranteed free for use)	Conventional memory
0x00080000	0x0009FBFF	approximately 120 KiB, depending on EBDA size	RAM (free for use, **if it exists**)	Conventional memory
0x0009FC00 (typical location)	0x0009FFFF	1 KiB	RAM (unusable)	EBDA (Extended BIOS Data Area)
0x000A0000	0x000FFFFF	384 KiB	various (unusable)	Video memory, ROM Area

BIOS Data Area (BDA)

The BDA is only partially standardized, and almost all the values stored there are completely obsolete and uninteresting. The following is a partial list. See the External Links references below for more detail.

address (size)	description
0x0400 (4 words)	IO ports for COM1-COM4 serial (each address is 1 word, zero if none)
0x0408 (3 words)	IO ports for LPT1-LPT3 parallel (each address is 1 word, zero if none)
0x040E (word)	EBDA base address >> 4 (**usually!**)
0x0410 (word)	packed bit flags for detected hardware
0x0417 (word)	keyboard state flags
0x041E (32 bytes)	keyboard buffer
0x0449 (byte)	Display Mode

0x044A (word)	number of columns in text mode
0x0463 (2 bytes, taken as a word)	base IO port for video
0x046C (word)	# of IRQ0 timer ticks since boot
0x0475 (byte)	# of hard disk drives detected
0x0480 (word)	keyboard buffer start
0x0482 (word)	keyboard buffer end
0x0497 (byte)	last keyboard LED/Shift key state

Extended BIOS Data Area (EBDA)

You may see "maps" of the EBDA if you search the web. However, those maps are for the original IBM BIOS EBDA. They do not apply to any current EBDA, used by any current BIOS. The EBDA area is not standardized. It **does** contain data that your OS will need, but you must do a bytewise pattern search to find those tables. (See Plug-and-Play.)

ROM Area

start	end	size	region/exception	description
Standard usage of the ROM Area				
0x000A0000	0x000BFFFF	128 KiB	video RAM	VGA display memory
0x000C0000	0x000C7FFF	32 KiB (typically)	ROM	Video BIOS
0x000C8000	0x000EFFFF	160 KiB (typically)	ROMs and unusable space	Mapped hardware & Misc.
0x000F0000	0x000FFFFF	64 KiB	ROM	Motherboard BIOS

"Upper" Memory (> 1 MiB)

The region of RAM above 1 MiB is not standardized, well-defined, or contiguous. There are likely to be regions of it that contain memory mapped hardware, that nothing but a device driver should ever access. There are likely to be regions of it that contain ACPI tables which your initialization code will probably want to read, and that then can be overwritten and reused. Some ACPI areas cannot be "reclaimed" this way. Some of the computer's RAM may extend above 4 GiB.

Use the BIOS function INT 15h, EAX=0xE820 to get a reliable map of Upper Memory.

start	end	size	region/exception	description
High Memory				

0x00100000	0x00EFFFFF	0x00E00000 (14 MiB)	RAM -- free for use (if it exists)	Extended memory [1, 2]
0x00F00000	0x00FFFFFF	0x00100000 (1 MiB)	Possible memory mapped hardware	ISA Memory Hole 15-16MB [3]
0x01000000	????????	???????? (whatever exists)	RAM -- free for use	More Extended memory [1]
0xC0000000 (sometimes, depends on motherboard and devices)	0xFFFFFFFF	0x40000000 (1 GiB)	various (typically reserved for memory mapped devices)	Memory mapped PCI devices, PnP NVRAM?, IO APIC/s, local APIC/s, BIOS, ...
0x0000000100000000 (possible memory above 4 GiB)	????????????????	???????????????? (whatever exists)	RAM -- free for use (PAE/64bit)	More Extended memory [1]
????????????????	????????????????	????????????????	Possible memory mapped hardware	Potentially usable for memory mapped PCI devices in modern hardware (but typically not, due to backward compatibility)

[1]: Different computers have different amounts of RAM, therefore the amount of extended memory you might find will vary and may be anything from "none" (e.g. an old 80386 system) to "lots".

[2]: Free for use except that your bootloader (ie. GRUB) may have loaded your "modules" here, and you don't want to overwrite those.

[3]: The "ISA Memory Hole" (from 0x00F00000 to 0x00FFFFFF) was used for memory mapped ISA devices (e.g. video cards). Modern computers have no need for this hole, but some chipsets still support it (as an optional feature) and some motherboards may still allow it to be enabled with BIOS options, so it may exist in a modern computers with no ISA devices.

See Also

- Detecting Memory (x86)

External Links

- http://www.nondot.org/sabre/os/files/Booting/BIOS_SEG.txt -- detailed BIOS Data Area map
- http://www.bioscentral.com/misc/bda.htm -- another detailed BIOS Data Area map
- Geezer's memory layout description (http://files.osdev.org/mirrors/geezer/osd/ram/index.htm#layout)
- http://stanislavs.org/helppc/bios_data_area.html

Retrieved from "http://wiki.osdev.org/index.php?title=Memory_Map_(x86)&oldid=16995"
Category: X86

- This page was last modified on 3 November 2014, at 11:15.
- This page has been accessed 203,557 times.