Global Descriptor Table

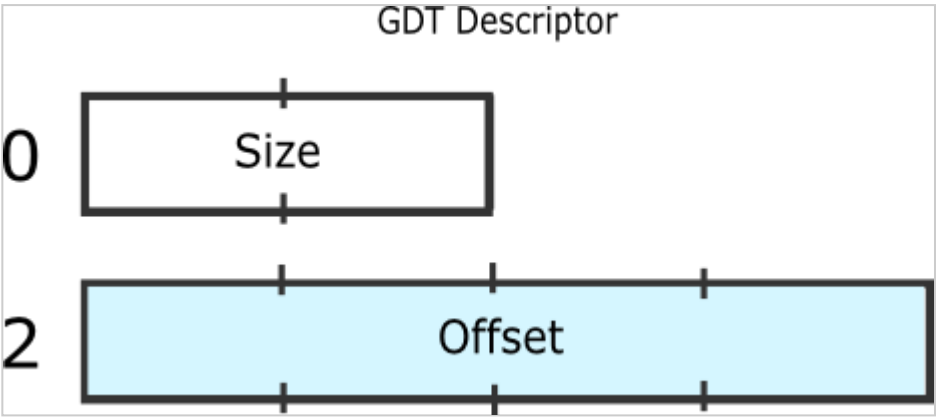
From OSDev Wiki

The **Global Descriptor Table** (**GDT**) is specific to the IA32 architecture. It contains entries telling the CPU about memory segments. A similar Interrupts Descriptor Table exists containing tasks and interrupts descriptors. Read the GDT Tutorial.

Contents

Structure

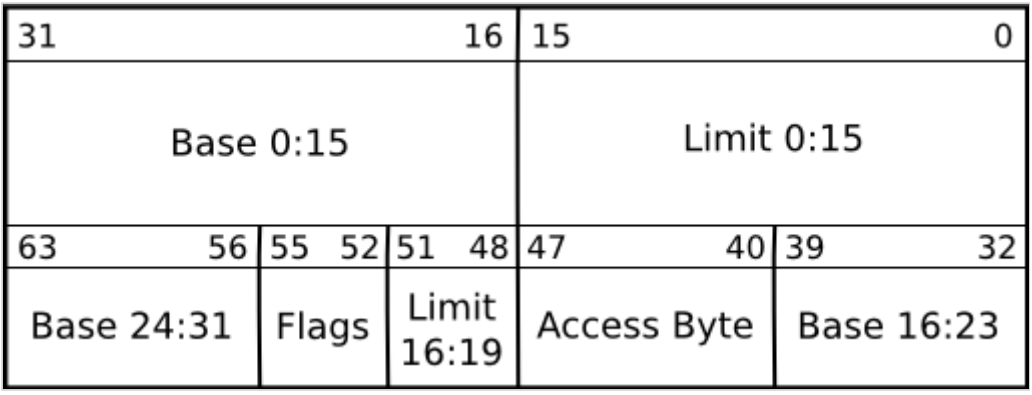
The GDT is loaded using the LGDT assembly instruction. It expects the location of a GDT description structure:



GDTR

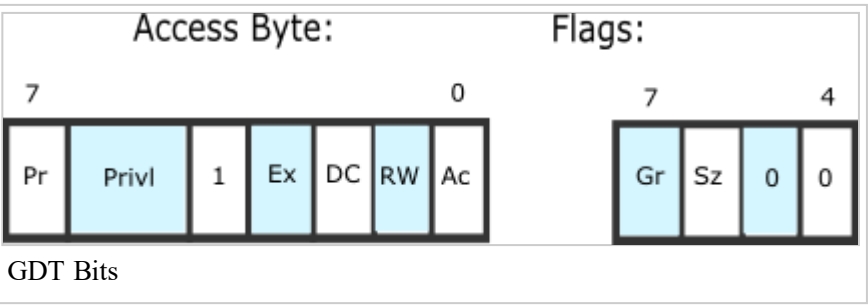
The `offset` is the linear address of the table itself, which means that paging applies. The `size` is the size of the table subtracted by 1. This is because the maximum value of `size` is 65535, while the GDT can be up to 65536 bytes (a maximum of 8192 entries). Further no GDT can have a size of 0.

The table contains 8-byte entries. Each entry has a complex structure:



A GDT Entry

What "Limit 0:15" means is that the field contains bits 0-15 of the `limit` value. The `base` is a 32 bit value containing the linear address where the segment begins. The `limit`, a 20 bit value, tells the maximum addressable unit (either in 1 byte units, or in pages). Hence, if you choose page granularity (4 KiB) and set the `limit` value to 0xFFFFF the segment will span the full 4 GiB address space. Here is the structure of the access byte and flags:



GDT Bits

The bit fields are:

- **Pr:** Present bit. This must be **1** for all valid selectors.
- **Privl:** Privilege, 2 bits. Contains the ring level, 0 = highest (kernel), 3 = lowest (user applications).
- **Ex:** Executable bit. If **1** code in this segment can be executed, ie. a code selector. If **0** it is a data selector.
- **DC:** Direction bit/Conforming bit.
  - Direction bit for data selectors: Tells the direction. **0** the segment grows up. **1** the segment grows down, ie. the `offset` has to be greater than the `limit`.
  - Conforming bit for code selectors:
    - If **1** code in this segment can be executed from an equal or lower privilege level. For example, code in ring 3 can far-jump to *conforming* code in a ring 2 segment. The `privl`-bits represent the highest privilege level that is allowed to execute the segment. For example, code in ring 0 cannot far-jump to a conforming code segment with `privl==0x2`, while code in ring 2 and 3 can. Note that the privilege level remains the same, ie. a far-jump form ring 3 to a `privl==2`-segment remains in ring 3 after the jump.
    - If **0** code in this segment can only be executed from the ring set in `privl`.
- **RW:** Readable bit/Writable bit.
  - Readable bit for code selectors: Whether read access for this segment is allowed. Write access is never allowed for code segments.
  - Writable bit for data selectors: Whether write access for this segment is allowed. Read access is always allowed for data segments.
- **Ac:** Accessed bit. Just set to **0**. The CPU sets this to **1** when the segment is accessed.
- **Gr:** Granularity bit. If **0** the `limit` is in 1 B blocks (byte granularity), if **1** the `limit` is in 4 KiB blocks (page granularity).
- **Sz:** Size bit. If **0** the selector defines 16 bit protected mode. If **1** it defines 32 bit protected mode. You can have both 16 bit and 32 bit selectors at once.

Not shown in the picture is the 'L' bit (bit 21, next to 'Sz') which is used for x86-64 mode. See Figure 3-8, "Segment Descriptor" of the Intel System Programmer's manual.

See also

Articles

- GDT Tutorial
- Segmentation
- LGDT

External references

- Protected Mode tutorial (http://files.osdev.org/mirrors/geezer/os/pm.htm)
- Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3A:. System Programming Guide, Part 1 (order number 253668) (http://www.intel.com/design/processor/manuals/253668.pdf) chapter 2.4