Global Descriptor Table

From OSDev Wiki

The **Global Descriptor Table** (**GDT**) is a binary data structure specific to the IA-32 and x86-64 architectures. It contains entries telling the CPU about memory segments. A similar Interrupt Descriptor Table exists containing task and interrupt descriptors.

It is recommended to read the GDT Tutorial.

Contents

GDTR

The **GDT** is pointed to by the value in the **GDTR** register. This is loaded using the **LGDT** assembly instruction, whose argument is a pointer to a **GDT Descriptor** structure:

GDT Descriptor (GDTR)

79 (64-bit mode) **48 (32-bit mode)** **16**	**15** **0**
Offset 63 (64-bit mode) 31 (32-bit mode) 0	**Size** 15 0

- **Size:** The size of the table in bytes subtracted by 1. This subtraction occurs because the maximum value of **Size** is 65535, while the **GDT** can be up to 65536 bytes in length (8192 entries). Further, no **GDT** can have a size of 0 bytes.
- **Offset:** The linear address of the **GDT** (not the physical address, paging applies).

Note that the amount of data loaded by **LGDT** differs in 32-bit and 64-bit modes, the offset is 4 bytes long in 32-bit mode and 8 bytes long in 64-bit mode.

For more information, see **Section 2.4.1: Global Descriptor Table Register (GDTR)** and **Figure 2-6: Memory Management Registers** of the Intel Software

Developer Manual, Volume 3-A.

Table

The entries in the **GDT** are 8 bytes long and form a table like this:

Global Descriptor Table

Address	Content
GDTR Offset + 0	Null
GDTR Offset + 8	Entry 1
GDTR Offset + 16	Entry 2
GDTR Offset + 24	Entry 3
...	**...**

The first entry in the **GDT** (Entry 0) should always be null and subsequent entries should be used instead.

Entries in the table are accessed by **Segment Selectors**, which are loaded into **Segmentation** registers either by assembly instructions or by hardware functions such as **Interrupts**.

Segment Descriptor

Each entry in the table has a complex structure:

Segment Descriptor

63 **56**	**55** **52**	**51** **48**	**47** **40**	**39** **32**
Base 31 24	**Flags** 3 0	**Limit** 19 16	**Access Byte** 7 0	**Base** 23 16
31 **16**			**15** **0**	
Base 15 0			**Limit** 15 0	

- **Base:** A 32-bit value containing the linear address where the segment begins.
- **Limit:** A 20-bit value, tells the maximum addressable unit, either in 1 byte units, or in 4KiB pages. Hence, if you choose page granularity and set the **Limit** value to 0xFFFFF the segment will span the full 4 GiB address space in 32-bit mode.

In 64-bit mode, the **Base** and **Limit** values are ignored, each descriptor covers the entire linear address space regardless of what they are set to.

For more information, see **Section 3.4.5: Segment Descriptors** and **Figure 3-8: Segment Descriptor** of the Intel Software Developer Manual, Volume 3-A.

Access Byte

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
P	**DPL**		**S**	**E**	**DC**	**RW**	**A**

- **P:** Present bit. Allows an entry to refer to a valid segment. Must be set (**1**) for

any valid segment.
- **DPL:** Descriptor privilege level field. Contains the CPU Privilege level of the segment. **0** = highest privilege (kernel), **3** = lowest privilege (user applications).
- **S:** Descriptor type bit. If clear (**0**) the descriptor defines a system segment (eg. a Task State Segment). If set (**1**) it defines a code or data segment.
- **E:** Executable bit. If clear (**0**) the descriptor defines a data segment. If set (**1**) it defines a code segment which can be executed from.
- **DC:** Direction bit/Conforming bit.
  - For data selectors: Direction bit. If clear (**0**) the segment grows up. If set (**1**) the segment grows down, ie. the **Offset** has to be greater than the **Limit**.
  - For code selectors: Conforming bit.
    - If clear (**0**) code in this segment can only be executed from the ring set in **DPL**.
    - If set (**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 **DPL** field 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 where **DPL** is 2, while code in ring 2 and 3 can. Note that the privilege level remains the same, ie. a far-jump from ring 3 to a segment with a **DPL** of 2 remains in ring 3 after the jump.
- **RW:** Readable bit/Writable bit.
  - For code segments: Readable bit. If clear (**0**), read access for this segment is not allowed. If set (**1**) read access is allowed. Write access is never allowed for code segments.
  - For data segments: Writeable bit. If clear (**0**), write access for this segment is not allowed. If set (**1**) write access is allowed. Read access is always allowed for data segments.
- **A:** Accessed bit. Best left clear (**0**), the CPU will set it when the segment is accessed.

Flags

3	2	1	0
G	**DB**	**L**	Reserved

- **G:** Granularity flag, indicates the size the **Limit** value is scaled by. If clear (**0**), the **Limit** is in 1 Byte blocks (byte granularity). If set (**1**), the **Limit** is in 4 KiB blocks (page granularity).
- **DB:** Size flag. If clear (**0**), the descriptor defines a 16-bit protected mode segment. If set (**1**) it defines a 32-bit protected mode segment. A GDT can have both 16-bit and 32-bit selectors at once.
- **L:** Long-mode code flag. If set (**1**), the descriptor defines a 64-bit code segment. When set, **DB** should always be clear. For any other type of segment (other code types or any data segment), it should be clear (**0**).

System Segment Descriptor

For system segments, such as those defining a **Task State Segment** or **Local Descriptor Table**, the format of the **Access Byte** differs slightly, in order to define different types of system segments rather than code and data segments.

For more information, see **Section 3.5: System Descriptor Types** and **Figure 3-2: System-Segment and Gate-Descriptor Types** of the Intel Software Developer Manual, Volume 3-A.

Access Byte

7	6	5	4	3	2	1	0
P	**DPL**		**S**	**Type**			

- **Type:** Type of system segment.

Types available in 32-bit protected mode:

- **0x1:** 16-bit TSS (Available)
- **0x2:** LDT
- **0x3:** 16-bit TSS (Busy)
- **0x9:** 32-bit TSS (Available)
- **0xB:** 32-bit TSS (Busy)

Types available in Long Mode:

- **0x2:** LDT
- **0x9:** 64-bit TSS (Available)
- **0xB:** 64-bit TSS (Busy)

Long Mode System Segment Descriptor

For a **Task State Segment** or **Local Descriptor Table** in **Long Mode**, the format of a **Segment Descriptor** differs to ensure that the **Base** value can contain a 64-bit **Linear Address**. It takes up the space in the table of two usual entries, in a little endian format, such that the lower half of this entry precedes the higher half in the table.

For more information, see **Section 7.2.3: TSS Descriptor in 64-bit Mode** and **Figure 7-4: Format of TSS and LDT Descriptors in 64-bit Mode** of the Intel Software Developer Manual, Volume 3-A.

64-bit System Segment Descriptor

127					**96**
Reserved					
95					**64**
Base 63					32
63 **56**	**55** **52**	**51** **48**	**47** **40**	**39**	**32**
Base 31 24	**Flags** 3 0	**Limit** 19 16	**Access Byte** 7 0	**Base** 23	16
31		**16**	**15**		**0**
Base 15		0	**Limit** 15		0

See Also

Articles

- GDT Tutorial
- Getting to Ring 3
- Segmentation
- http://www.osdever.net/tutorials/view/the-world-of-protected-mode - how to set up GDT in assembler

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

Category: X86 CPU
