

The Segment Descriptor Cache

By Robert R. Collins

It is easy to underestimate the importance of something when you don't know what it is or how it works. As early as the 80286, all Intel x86 processors have included an entity called the "segment-descriptor cache" which works behind the scenes, hidden from you. It is updated each time a segment register is loaded. It is used for all memory accesses by all Intel x86 processors since the 80286. If you're an end user, you've probably used programs that depend upon the functions of the segment-descriptor cache. If you're an engineer, there is a high probability that you've relied upon the functions of the segment-descriptor cache – and you might not have realized it. If you're an engineer who writes any low-level code, programs hardware, or programs in protected mode, then you should be aware of the segment-descriptor cache and how it works.

From the 80286 to 80486, the meaning of "segment-descriptor cache" was unambiguous, referring to an internal microprocessor structure that stores the internal representation of the segment registers. This representation includes the segment base address, limit, and access rights. With the Pentium, Intel introduced a 94-entry, two-way set associative cache of segment-descriptor cache entries. Therefore, the phrase "segment-descriptor cache" is now ambiguous, with two possible meanings. Making matters worse, the new segment-descriptor cache was removed from the Pentium Pro design, but reintroduced in the Pentium II. (The lack of the new segment-descriptor cache in the Pentium Pro largely accounted for its poor 16-bit performance.) In this column, I'll discuss the original segment-descriptor cache that has existed since the 80286 (and remains in all modern Intel x86 processors) and the role of the segment-descriptor cache in microprocessor memory management.

Loading Descriptor Cache Registers

Whether in real, protected, virtual-8086, or system-management mode, the microprocessor stores the base address of each segment in a hidden descriptor-cache registers. Each time a segment register is loaded, the segment-base address, segment-size limit, and segment-access attributes (access rights) are loaded (cached) into these hidden registers. To enhance performance, subsequent memory references are made via the descriptor-cache registers. Without this optimization, each memory access would require the microprocessor to perform many time-consuming tasks. In real mode, the microprocessor would need to calculate the physical address from the segment register value. The access rights would always indicate a read/write data segment (even for the code segment). The limit would always be 64 KB of memory. In protected mode, the segment base must be looked up in the appropriate descriptor table. The segment base is composed of a combination of fields in the descriptor table. The segment access rights and segment limit are also contained in the descriptor table. The microprocessor would need to access those structures for each memory access. These descriptor-table values reside in memory, where accesses tend to be slow when compared to accesses within the microprocessor. Therefore, without an internal segment-descriptor cache to cache these values, each memory access would implicitly require many other accesses to memory.

Now consider the differences between real mode and protected mode. If the segment descriptor cache didn't exist, determining segment base, limit, and access rights would require more than one CPU cycle to complete. Therefore, the segment-descriptor cache exists to eliminate these potential deficiencies. It exists to allow all of these differences to be resolved at the time each segment is loaded. The performance penalty is incurred only once. Thereafter, all memory management is performed according to the values in the segment descriptor caches for each respective segment register.

At power-up, the descriptor-cache registers are loaded with fixed, default values – the CPU is in real mode, and all segments are marked as read/write data segments, including the code segment (CS). According to Intel, each time any segment register is loaded in real mode, the base address is calculated as 16 times the segment value, while the access rights and size limit attributes are given fixed, "real-mode compatible" values. This is not true. In fact, only the CS descriptor caches for the 286, 386, and 486 get loaded with fixed values each time the segment register is loaded. Loading CS, or any other segment register in real mode, on later Intel processors doesn't change the access rights or the segment size limit attributes stored in the descriptor cache registers. For these segments, the access rights and segment size limit attributes from any previous setting are honored. Thus, it is possible to have a four-GB read-only data segment in real mode on the 80386 – but Intel won't acknowledge this mode of operation, though it is implicitly supported. Furthermore, Intel can't remove it without rendering many software programs ineffective.

Protected mode differs from real mode in this respect: As each time a segment register is loaded, the descriptor cache register gets fully loaded; no values are honored. The descriptor cache is loaded directly from the descriptor table. The CPU checks the validity of the segment by testing the access rights in the descriptor

table. Complete checks are made, and illegal values will generate exceptions. Any attempt to load CS with a read/write data segment will generate a protection error. Likewise, any attempt to load a data segment register as an executable segment will also generate an exception. The CPU strictly enforces these protection rules. If the descriptor-table entry passes all the tests, then the descriptor-cache register gets loaded.

Format of Descriptor Cache Registers

The layout of the segment-descriptor cache registers change with almost every processor generation, though their functions do not. These differences are known as "implementation specific" because their exact layout and contents depend on the design and implementation of the microprocessor. For the most part, the fields of the segment-descriptor cache mirror the fields in the protected mode descriptor table. For 32-bit descriptor-table entries, the segment-base address, segment-access rights, and segment-limit fields are not contiguous. These related fields are combined before being put in the segment-descriptor cache. [Figure 1](#) shows the relationship between fields in the descriptor table and segment-descriptor cache.

Figure 1: Combining fields from the descriptor table into the segment-descriptor cache.

Offset	63..56	55	54	53	52	51..48	47	46..45	44	43..40	39..16	15..00
Description	Base[31:24]	G	D/B	0	AVL	Limit[19:16]	P	DPL	S	Type	Base[23:00]	Limit[15:00]

Segment Base

Segment Access Rights

Segment Limit

Segment-Descriptor Cache

It is useful to know the layout of the fields inside the segment-descriptor cache. The segment base and segment limit are always combined from the descriptor table to form a complete base address and segment limit inside the segment-descriptor cache. The format of the access rights within the segment-descriptor cache changes from implementation to implementation. Likewise, the order of the fields within the cache can change. Regardless, knowing the format of the segment-descriptor cache can make you more productive, reducing both development and debugging time. [Tables 1 through 4](#) show the descriptor cache entry format for all Intel x86 processors from the 80286 through Pentium Pro.

Table 1 - 80286 Descriptor Cache Entry

Bit Position	47..32	31	30..29	28	27..24	23..00
Description	16-bit Limit	P	DPL	S	Type	24-bit Base

Table 2 - 80386 and 80486 Descriptor Cache Entry

Bit Position	95..64	63..32	31..24	23	22..21	20	19..16	15	14	13..0
Description	32-bit Limit	32-bit Base	0	P	DPL	S	Type	0	D / B	0

Table 3 - Pentium Descriptor Cache Entry

Bit Position	95..79	78	77..72	71	70..69	68	67..64	63..32	31..00
Description	0	D/B	0	P	DPL	S	Type	32-bit Base	32-bit Limit

Table 4 - Pentium Pro Descriptor Cache Entry

Bit Position	95..64	63..32	31	30	29..24	23	22..21	20	19..16	15..00
Description	32-bit Base	32-bit Limit	0	D/B	0	P	DPL	S	Type	Segment Selector

Descriptor-Cache Registers In Real Life

There are different ways to take advantage of the segment-descriptor cache registers. System-management mode (SMM) gives you direct control over each field in the segment-descriptor cache. (See my [DDJ January/March/May 1997](#) columns for an in-depth look at System Management Mode.) In-circuit emulators (ICEs) also allow direct control over each field in the segment descriptor cache. (Refer to my [DDJ July/September/November 1997](#) columns for information on in-circuit emulation.)

For instance, when writing any low-level assembly-language programs (such as OS kernels, device drivers, BIOS, or protected-mode programming), I make common, simple mistakes. I sometimes make a mistake when creating my segment descriptor table, usually the Global Descriptor Table (GDT). I may have created the GDT using an incorrect base address, segment limit, or access rights. Ultimately, my program fails, and I must use the ICE as a debugging tool. I'll then insert the undocumented ICEBP instruction into my code to instruct the ICE to breakpoint at the suspected point of failure (see <http://www.x86.org/secrets/opcodes/ICEBP.html>). Within moments, I discover that I used incorrect values in building the descriptor table. Using the ICE, I can load each field of the segment-descriptor cache. If I used an incorrect segment base address, I can correct it and continue. Likewise, I can make the same corrections for the segment limit and segment access rights. I know that these values are "sticky," meaning that they don't get changed until a new segment register value is loaded. Therefore, I can make these changes, and continue debugging my program. Using this technique, I can usually discover six or more bugs in my program before recompiling. Because I don't need to recompile my program after discovering each and every mistake, I save valuable development and debugging time.

Programming in SMM implicitly takes advantage of the segment-descriptor cache registers. The segment-descriptor cache registers are saved and restored along with the remaining microprocessor state in the SMM state save map. These values are saved and restored upon entry and exit to system-management mode. In my [March 1997 DDJ column](#), I disclosed all of the undocumented fields (known as "reserved" fields in Intel parlance) in the Pentium SMM state save map. As I discussed, the segment-descriptor cache registers are stored in these reserved fields.

It is possible to manipulate these segment-descriptor cache values from within the SMM handler. The segment base may be changed to a value that is inconsistent with its associated segment register value. The segment access rights may be manipulated to give current-privilege-level-3 (CPL-3) tasks CPL-0 access (an obvious breach of security). The segment limit may be changed to create a segment with a four-GB limit while in real mode. Using SMM, it is possible to change the segment attributes to values that are programmatically impossible; for example, a real-mode segment at two MB, a segment limit size of 4-gigabytes minus 16, or a read/write code segment in protected mode (not to mention CPL-0 access within a CPL-3 task).

Descriptor Cache Anomalies and creating "Unreal Mode"

Using either of these methods to manipulate segment-descriptor caches can be challenging. However, there's another programatic way of putting segment-descriptor caches to work – creating a CPU operating mode known as "unreal" mode.

Unreal mode is created when a real-mode segment has a four-GB segment limit. Unreal mode can be created without any hardware debuggers or SMM programming with a simple assembly-language program. Imagine a program that begins in real mode then transitions into protected mode. Once in protected mode, the program loads all of the segment registers with descriptors containing four-GB segment limits. After setting the segment limits, return immediately to real mode without restoring the segment registers to segments containing 64-KB segments (real-mode-compatible segments). Once in real mode, the segment limits will always retain their four-GB limits. Thereafter, DOS programs can take advantage of the entire 32-bit address space without resorting to protected-mode programming.

Unreal mode has been used commonly since it was discovered on the 80386. Unreal mode is so commonly used, in fact, that Intel has been forced to support this mode as part of legacy 80x86 behavior, though it's never been documented. Memory managers and games often take advantage of unreal mode. Source code that demonstrates how you can create unreal mode is available electronically from [DDJ](#) (see "Resource Center,") or at <ftp://ftp.x86.org/downloads/UNREAL.ZIP>.

The real-mode code segment (CS) descriptor cache behavior has changed between generations of Intel processors. The role of the code segment-descriptor cache in real mode differs between the 80286, 80386, and 80486 and all later Intel microprocessors: The earlier microprocessors honor the real-mode segment access rights in real mode until a far control transfer occurs; later processors ignore any access rights in the CS descriptor cache irrespective of far control transfers. On the earlier processors, any far control transfer set the CS descriptor cache access rights to its real-mode compatible value as a read/write data segment (value=0x93). Later processors leave the original value intact, but ignore its contents. Therefore, transitions from real to protected mode on the later processors immediately causes the behavior to revert to its stagnant

CS descriptor-cache access rights value. On earlier processors, the CS limit is also restored to its real-mode compatible value (64 KB). Later processors leave the CS segment limit alone, making its behavior consistent with the other data segment registers.

From the 80286 to the Pentium, all Intel processors derive their current privilege level (CPL) from the SS access rights. The CPL is loaded from the SS descriptor table entry when the SS register is loaded. The undocumented LOADALL instruction (or system-management mode RSM instruction) can be used to manipulate the SS descriptor-cache access rights, thereby directly manipulating the CPL of the microprocessors. (See <http://www.x86.org/articles/loadall/> for a description of LOADALL.) The Pentium Pro behaves differently: Once the CPL is loaded into the Pentium Pro, it is not internally derived from the SS access rights. The Pentium Pro retains a separate CPL register. Through the system-management mode RSM instruction, you can directly manipulate the CPL of the Pentium Pro, though not by manipulating the SS access rights value. (I will discuss the Pentium Pro SMM state save map and all of the secrets contained therein in a future column.)

Conclusion

I use the segment-descriptor cache registers every day – when I’m debugging on my ICE to help correct common protected-mode programming errors, programming in system-management mode to create events, or creating real-mode segments that can address the entire four-GB address space. The use of the segment-descriptor cache is highly implementation specific, meaning the behavior and layout of the segment-descriptor cache is dependant upon the implementation of the specific microprocessor. Intel doesn’t guarantee that the behavior of the descriptor cache will remain the same from microprocessor to microprocessor. Therefore, it would be foolhardy to write any production-quality source code which depends upon this behavior (except unreal mode).

[Back to Dr. Dobb's Undocumented Corner home page](#)