1 Intel Memory Management

1.1 Segments

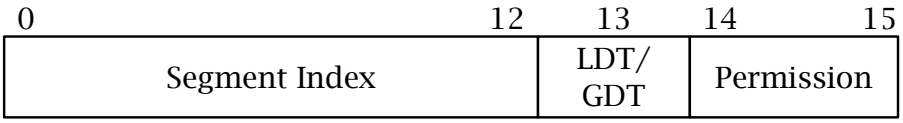
Intel began to integrate virtual memory into its commodity processors with the 286, which was a segmented architecture. On the 286, the segment registers just held the base address of a segment. On the 386, the segment registers held a *segment selector*. The basic idea was that the processor contains segment selectors, which are indices into a segment table. When a process uses an address (through one of the hundreds of intel instructions), the appropriate segment selector(s) are read and used to index into the segment table. That entry is then used to generate the actual physical address. There are two segment tables. The **LDT** is the local descriptor table and is process-local. The **GDT** is the global descriptor table and is meant to store addresses shared by multiple processes.

The processor has 6 registers that hold segment descriptors:

- **CS: code segment** contains the descriptor for the code segment. This is used to address instruction memory.
- **DS: data segment** contains the descriptor for program data.
- **SS: stack segment** contains the descriptor for the program stack.
- **ES, FS, GS** are additional 'miscellaneous' segments.

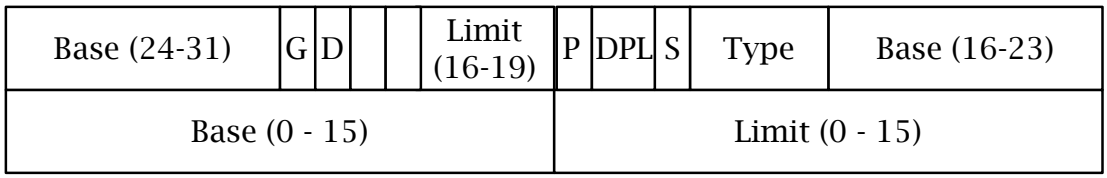
A segment selector itself is more than just an index. In addition to the 13 bit index (into a table that contains 8192 entries), there's an LDT/GDT flag (1 bit), and a permissions field (2 bits). The permission field (actually known as the Requested Privledge Level or RPL) specifies the 'ring' that the segment applies to. The actual privledge check is fairly complex, involving the processor's current privledge level, and an additional privledge level specified in the table entry.

The segment table



1.2 Segment Descriptors

Segment descriptors are the entries inside the segment table. Each descriptor is 8 bytes long, and is a fairly complicated hunk of data. Essentially, a descriptor is just a 32-bit base address, a segment limit (20 bits), and some flags and permissions data. However, because these descriptors apply to both 32 and 16 bit systems, the values are split into 16 and 8 bit chunks (for historical reasons).



Here's a brief description of the other fields:

- **G:** Granularity. There are two options, bytes(0) or pages(1)
- **D:** 16 or 32-bit segment flag
- **P:** Presence. Whether the segment is there or not (used for systems that swap memory on a segment basis).
- **DPL:** Privledge level for the segment, RPL or CPL must exceed this for the request to be granted.
- **S:** System(0) or application(1) flag.

1.3 Using the Segment Selectors

The x86 is a CISC architecture. In a nutshell, this means that the x86 instruction set is designed to make the job of being an assembly programmer easier at the expense of making the job of a compiler writer considerably more difficult. A side-effect of the CISC-iness is that many of the instructions on the x86 have an implicit use of segment registers. For example the `push` and `pop` instruction implicitly use and update the `SS` register. Additionally, registers can be used explicitly (as in RISC architectures), for example:

```
movl $42, %fs:(%eax)
```

This instruction moves (as a 32-bit value) the immediate 42 into the address specified by adding the contents of `eax` to the base of the segment selected by `fs`.

1.4 Paging

As you can imagine, on the x86, paging is a touch more compicated. Segment descriptors add an extra computation. First, the 32-bit address is added to the segment base to generate a *linear address*. This linear address is then used as the virtual address for paging purposes (assuming it passes the limit check). The page tables are 2 level, with each level indexed by 10 bits. The page table entries themselves have the 20 bit frame number as well as bits for: Access, dirty, protection, and 'utility'. The TLB caches the entire page table entry (which intel calls the page descriptor), so that changes to these meta-data bits are fast.

2 Intel Extras

The x86 series of processors has had a long-standing and torrid love-affair with mode bits. A mode bit is basically architecture speak for a single processor-wide flag that affects the behavior of entire sets of instructions. For example, there are mode bits for operating in 8, 16 or 32 bit modes. Of interest to this discussion, there are mode bits for enabling/disabling paging and segmentation. Additionally, the intel distinguishes between 4 protection levels (although many systems only make use of two of them (user(3) and kernel(0))).

2.1 Physical Address Extension

PAE is one of the nifty new extension that Intel has added to its architectures recently to deal with physical memories larger than 4GB. When PAE is enabled, page descriptors inflate to 8 bytes, and frame numbers can be 24 bits large. Additionally, PAE permits another level of paging, however this level (refered to as the page directory pointer table) only has 4 entries. This level can be used or not, depending on the PS flag. Also, PAE finally adds an execution protection flag to x86 virtual memory. The so called `Nx` flag brings x86 virtual memory in line with what RISC systems have been doing for decades.

3 Windows Virtual Memory

Windows, since NT has been a 32-bit paged operating system. Windows simply partitions the 4GB virtual address space into 2GB of system (at the low addresses) and 2GB of application (in the high addresses), with a small amount of addresses (on the boundaries) reserved for invalid pointers (debugging). Allocation makes use of three page states:

- **Available** totally unused
- **Reserved** Unused, but claimed by a process
- **Committed** Allocated and used by a process

Windows processes have a local address space. It is impossible for processes to steal frames from other processes, which helps obviate starvation, but may lead to increased page faults. The size of the local area is assigned at process creation time and adjusted later. Essentially, if there's plenty of memory, the local area is increased. If memory is scarce, process' local memories are scavenged for unused pages. Additionally, because Microsoft has total control over the entire system software stack, they can make use of extensive hinting from upper software layers. For example, the GUI informs the memory manager when the user has minimized an application. This can be handy, presumably the user minimized the application because they're not planning on using it in the short term. Therefore, the memory manager can swap out all that application's pages without incurring too high a performance penalty.

4 Final Review

The final is structured into three sections: Processes, Threads, and Virtual Memory. Each section begins with a set of terms that you need to define in a few sentances. Afterwards, there are a few more substantial problems.

4.1 Processes

First, some terms:

- **Process**
- **Address Space**
- **Process Stack**
- **Process Heap**
- **Context Switch**
- **Fork**
- **Exec**
- **Round Robin**
- **Priority**
- **Blocked Process**
- **Zombie Process**
- **Signal Handler**

4.2 Threads

- **Thread**
- **Atomic Action**
- **Condition Variable**
- **Critical Section**
- **Semaphore**
- **Message Queues**

- **Mutual Exclusion**
- **Deadlock**
- **Busy Waiting/Spinning**

4.3 Virtual Memory

- **Segment**
- **Page**
- **Page Table**
- **Base**
- **MMU**
- **Limit**
- **Memory Protection**
- **Virtual Address**
- **TLB**