Segmentation

A program is divided into a number of segments each composed of independent address spaces.

A segment is a linear sequence of addresses from 0 to some maximum.

Segments can change dynamically in size over this range, e.g. stack segment increases for pushes and decreases for pops.

This allows individual segments to grow and shrink without running into each other.

Segments are logical entities, that the programmer **is aware of** and uses. Unlike paging, which is invisible to the user.

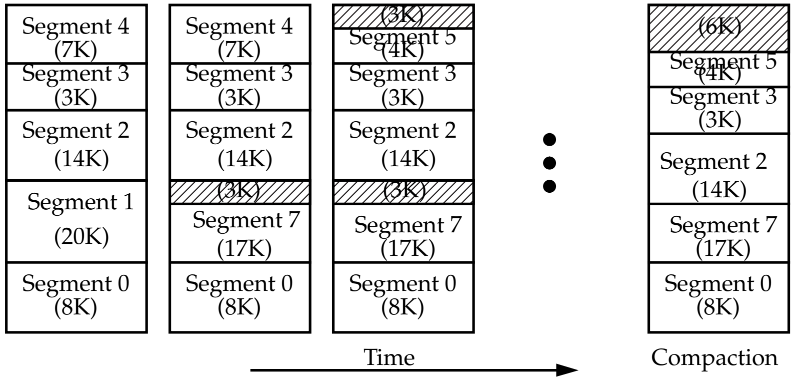
Segmentation Advantages:

- Programming and modularity: simplify the handling of data structures that are growing and shrinking.
- Programming and modularity: procedures can each be placed in a separate segment, allowing them to be changed without requiring recompilation of other procedures.
- Sharing: **shared libraries** allow sharing of procedures and data between several processes, e.g., procedures to support a GUI.
- Protection: different segments can have different forms of protection, e.g, a procedure segment can be specified as execute only and an integer array as read/write but not execute.

Pure Segmentation:

Pages are fixed in size, segments are variable sized.

Problem:



Segment 1 is evicted and segment 7 is put in its place.

Creates a 3K hole - external fragmentation.

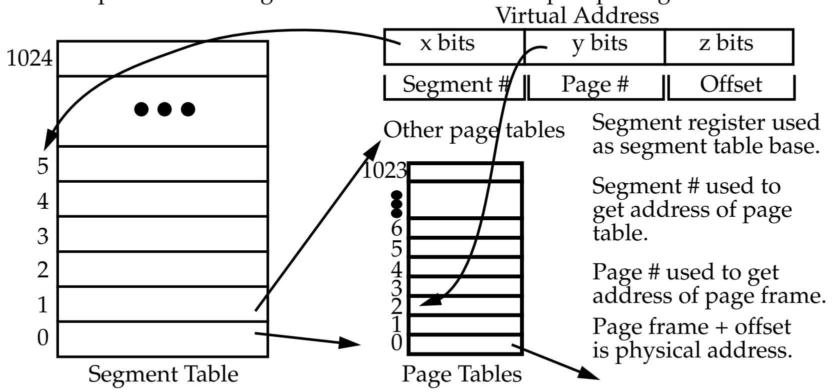
We have seen this before and used memory compaction to fix.

Segmentation with Paging:

Segments can be very large and it may be impossible to keep them in physical memory in their entirety.

Paging can be used with segmentation to solve this problem.

Each process has a segment table with one descriptor per segment:



Segment tables, since they can be large, may also be paged.

Address translation:

- Segment base register used to locate page table containing segment table.
- The segment number is used to locate the segment descriptor.
- If segment's page table (described by the segment descriptor) is in memory, the page number is used to locate the page descriptor using the base address given by the segment descriptor.
- If not, a segment fault occurs.
- The page descriptor is examined. If the page is present in memory, the page frame is combined with the offset to yield the physical address.
- If not, a page fault occurs.

TLBs are used to make this more efficient.

The Intel Pentium:

Protected mode operation: Segmentation

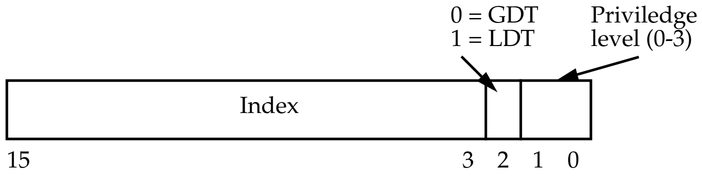
The Pentium allows up to 16K independent segments, each holding up to 1 billion 32-bit words.

Two tables used in address translation on the Pentium, LDT (Local Descriptor Table) and GDT (Global Descriptor Table).

Each process has its own LDT table but there is a single GDT, shared by all processes.

The LDT describes segments local to each program, e.g., code, data, stack, etc. while the GDT describes system segments, including the OS.

Segment access:



Pentium has 6 segment registers, CS, DS, ES, FS, GS and SS.

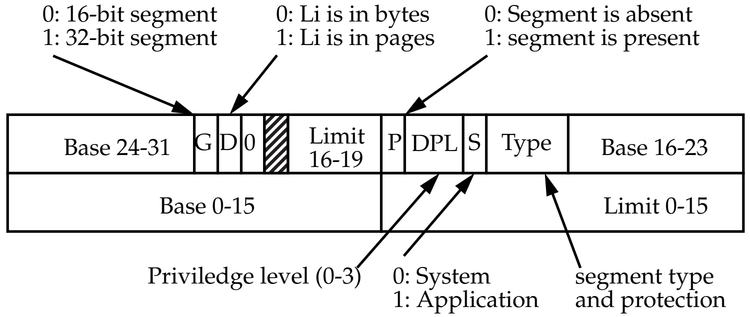
To access a segment, process loads a **selector** into one of the segment registers.

During execution, CS holds selector for code segment, DS holds selector for data segment and SS for stack segment.

GDT/LDT bit indicates whether the segment is local or global.
13 bits used to specify **segment descriptor** - up to 8K entries per table.
2 bits set priviledge level.

When the selector is loaded, the corresponding descriptor is fetched from the LDT or GDT (segment tables in memory) and stored in internal scratch registers for quick access.

Each **segment descriptor** consists of 8 bytes (64bits):



Low order 3 bits of selector cleared and added to LDTR or GDTR (special base registers pointing to segment tables) to generate physical address.

e.g., selector 72 (1001000) refers to entry 9 in the GDT which is located at address GDTR + 72.

Address translation:

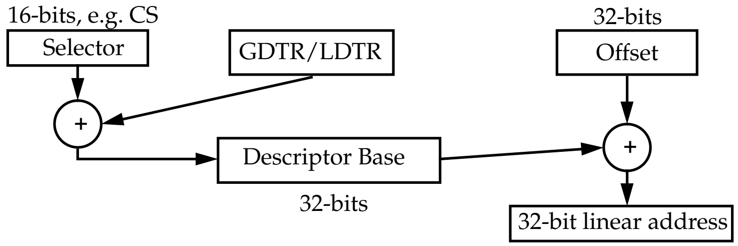
Process provides a (selector, offset) pair.

If the selector is currently paged out, a trap occurs.

Offset is then checked against the **Limit** in descriptor. If greater, a trap occurs.

Note: Only 20 bits is given by **Limit** field instead of 32. **G** bit (granularity) used to interpret the **Limit** value: If 0, **Limit** field is the exact segment size, up to 1 MB. If 1, then **Limit** specifies size in pages. Page size on the Pentium is always 4K (12 bits). 20+12 = 32.

If segment is in memory and offset is in range, Pentium adds the 32-bit **Base** field in the descriptor to the offset to form a l**inear address**.

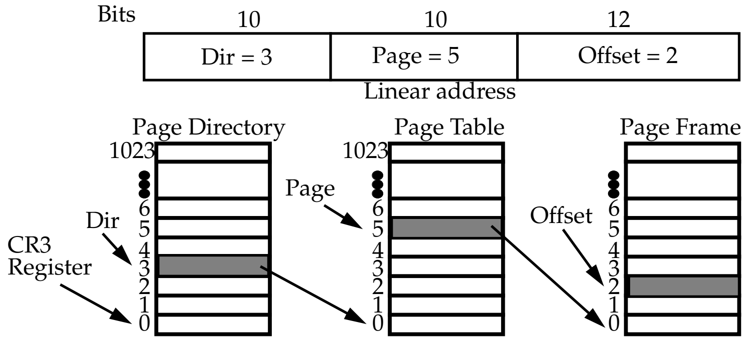


Pentium Paging:

If paging is disabled, linear address interpreted as a physical address.

With paging enabled, the linear address translated through a two-level page table (since 4K pages are used in a 32-bit address space = 1 million pages possible).

Each process has a **page directory** consisting of 1024 32-bit entries.



Page table entries are 32 bits each, 20 of which contain a page frame number.

The remaining 12 bits contain access and dirty bits.

Each page table contains 1024 4K page frames = 4 MB of memory.