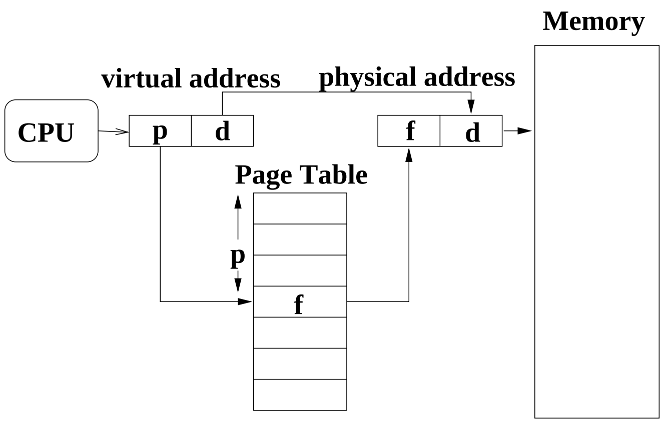
Paging — Last Time

- Process generates virtual addresses from 0 to Max.
- OS divides the process onto pages; manages a page table for every process; and manages the pages in memory (we have not covered this detail yet...)
- Hardware maps from virtual addresses to physical addresses.



Segmentation — Today

Segments take the user's view of the program and gives it to the OS.

- User views the program in logical *segments*, e.g., code, global variables, stack, heap (dynamic data structures), not a single linear array of bytes.
- Using this organization, the compiler generates references that identify the segment and the offset in the segment, e.g., a code segment with offset = 399 (which corresponds to procedure y).
- The processes thus use virtual addresses that are segments and segment offsets.

$\Longrightarrow$ Segments make it easier for the call stack and heap to grow dynamically. Why?

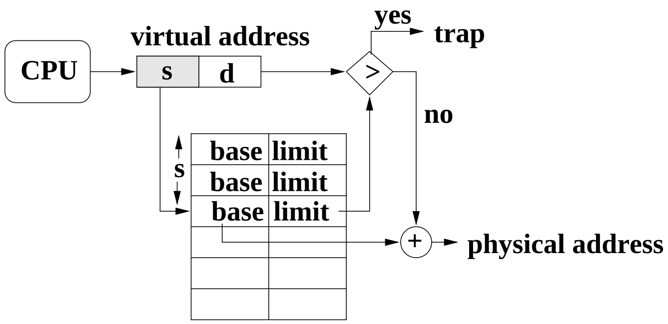
$\Longrightarrow$ Segments make sharing easier. Why?

$\Longrightarrow$ Segments make protection easier. Why?

Implementation of Segmentation:

- Segment table: each entry contains a base address in memory, length of segment, and protection information (can this segment be shared, read, modified, etc.).
- Hardware support: multiple base/limit registers.

Implementation of Segmentation

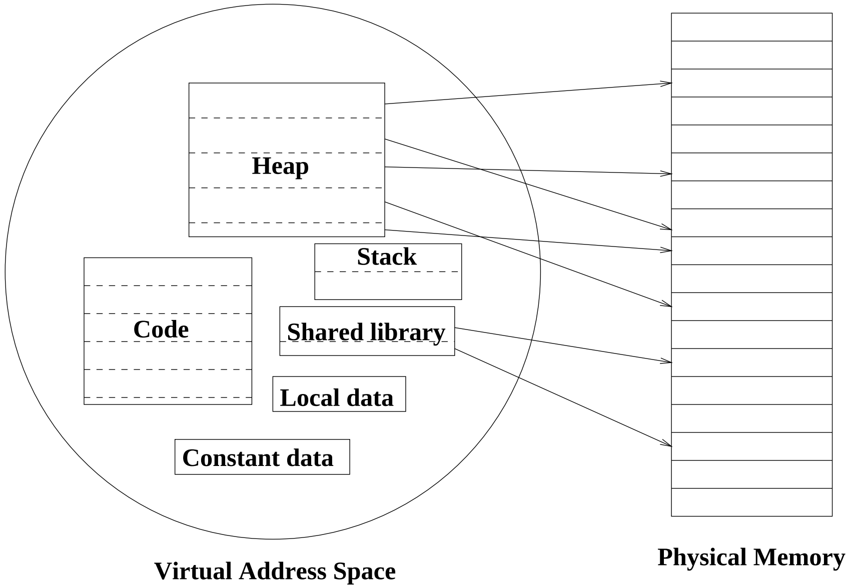


Issues:

- Compiler's job is a little more interesting, since it must generate virtual addresses whose upper order bits are a segment number.
- Segmentation can be combined with a dynamic or static relocation system, with each segment being allocated a contiguous piece of physical memory. External fragmentation can be a problem again.
- Similar memory mapping algorithm as paging. We need something like the TLB if programs can have lots of segments.

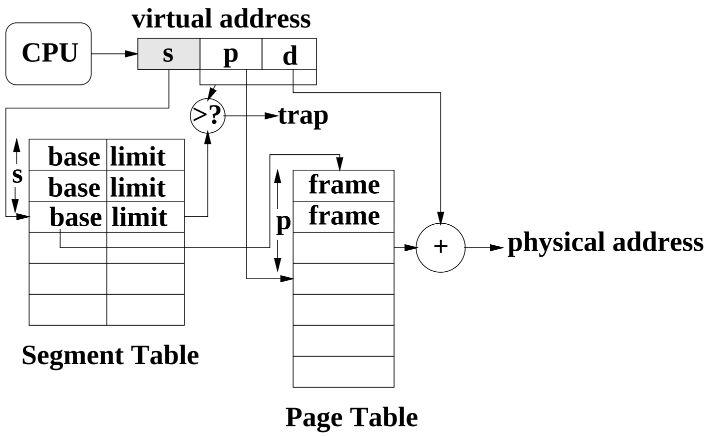
Let's combine the ease of sharing we get from segments with efficient memory utilization we get from pages.

Combining Segments and Paging



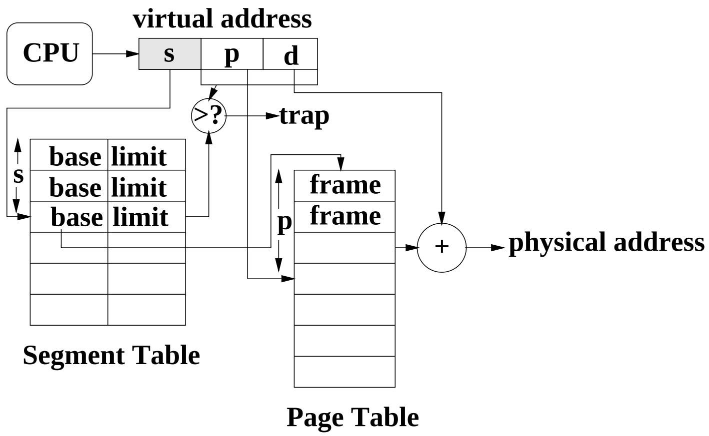
- Treat virtual address space as a collection of segments (logical units) of arbitrary sizes.
- Treat physical memory as a sequence of fixed size page frames.
- Segments are typically larger than page frames, so we map a logical segment onto multiple page frames by paging the segments.

Addresses in a Segmented Paging System



- A virtual address becomes a segment number, a page within that segment, and an offset within the page. (The segment number has to be generated separately by the compiler, but the compiler can treat the page number and offset simply as an offset within the segment.)
- The segment number indexes into the segment table which yields the base address of the page table for that segment.
- Check the remainder of the address (page number and offset) against the limit of the segment.
- Use the page number to index the page table. The entry is the frame. (The rest of this is just like paging.)
- Add the frame and the offset to get the physical address.

Combining Segments and Paging



- Given a memory size of 256 addressable words,
- a page table indexing 8 pages,
- a page size of 32 words, and
- 8 logical segments

1. How many bits is a physical address?

2. How many bits is a virtual address?

3. How many segment table entries do we need?

Combine Segments and Paging

Sharing

1. Share individual pages by copying page table entries.
2. Share whole segments by sharing segment table entries, which is the same as sharing the page table for that segment.
3. Need protection bits to specify and enforce read/write permission.
   - When would segments containing code be shared?
   - When would segments containing data be shared?

Implementation

- Where are the segment table and page tables stored?
  1. Can store segment tables in a small number of associative registers. Compilers then generate code to that number of segments. Page tables are in main memory with a TLB in hardware.
  2. OR Both the segment tables and page tables can be in main memory with the segment index and page index combined used in the TLB lookup. Number of segments is not limited by the hardware in this case.

$\Longrightarrow$ If the tables get too big, we can use multiple levels of paging and segmentation.

Combining Segments and Paging

- Protection and valid bits can go either on the segment or the page table entries.
- In a pure paging system (that is, one without segmentation), a page table contains (virtual address space)/(page size) entries. How many entries are required in a segmented paging system?
- Pages are used as a convenience to the OS
  1. OS can break up and scatter the memory of a process.
  2. OS can leave unused pages on disk.
  3. OS can manage memory in fixed size "pages".

  Benefits: faster process start times, faster process growth, memory sharing between processes.

  Costs: somewhat slower context switches, slower address translation.
- What is the performance of address translation of segmented paging compared to contiguous allocation with relocation?

  Compared to pure paging?
- How does fragmentation of segmented paging compare with contiguous allocation?

  With pure paging?

Putting it all together

Base & Bounds: simple, but inflexible

Segmentation:

- compiler's view presented to OS
- segment tables tend to be small
- memory allocation is expensive and complicated (first fit, worst fit, best fit).
- compaction is needed to resolve external fragmentation.

Paging:

- simplifies memory allocation since any page can be allocated to any frame
- page tables can be very large (especially when virtual address space is large and pages are small)

Segmentation & Paging

- only allocate the page table entries as the process is using (large virtual address spaces are not a problem, more on this topic later).
- easy memory allocation, any frame can be used
- sharing at either the page or segment level
- increased internal fragmentation over paging
- two lookups per memory reference

Virtual Memory — Next Time

Paging on demand for virtual address space larger than physical memory