Operating Systems

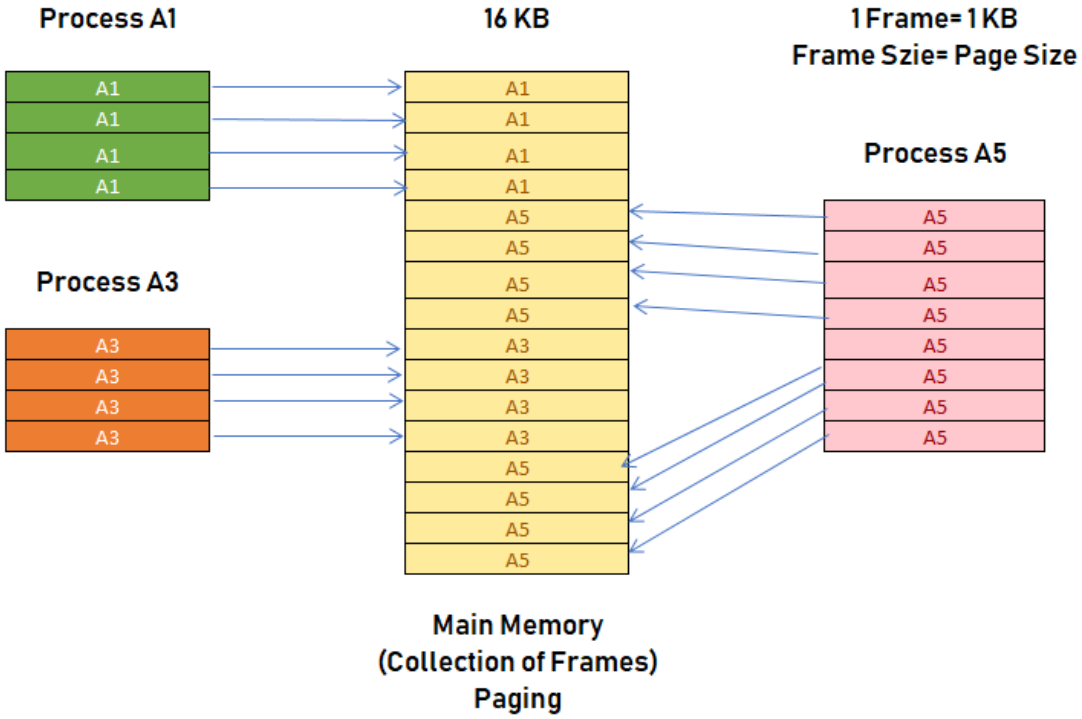
Isfahan University of Technology
Electrical and Computer Engineering Department
1400-1 semester

Zeinab Zali

Session 21: Main Memory: paging

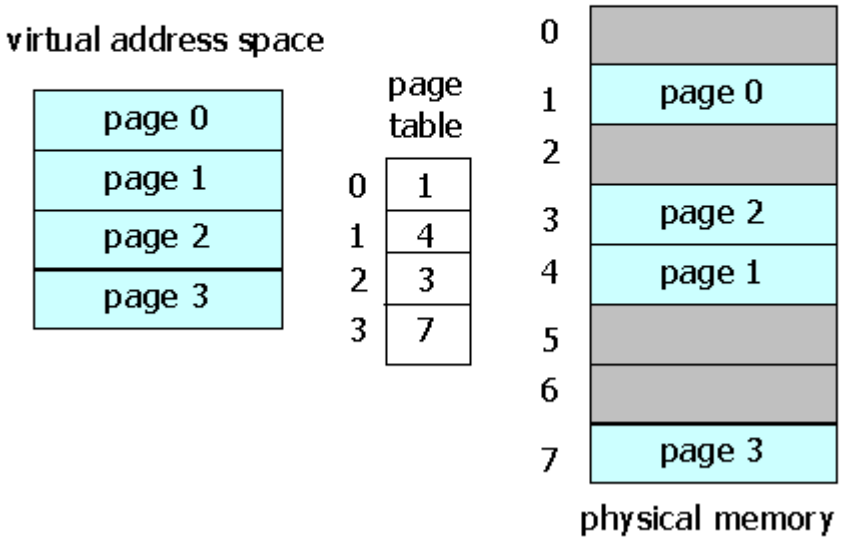
Paging

- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**



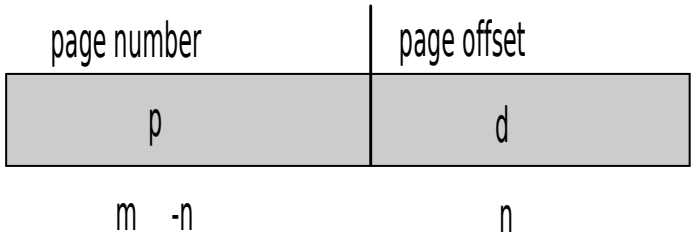
Paging

- Keep track of all free frames
- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- **Still have Internal fragmentation**



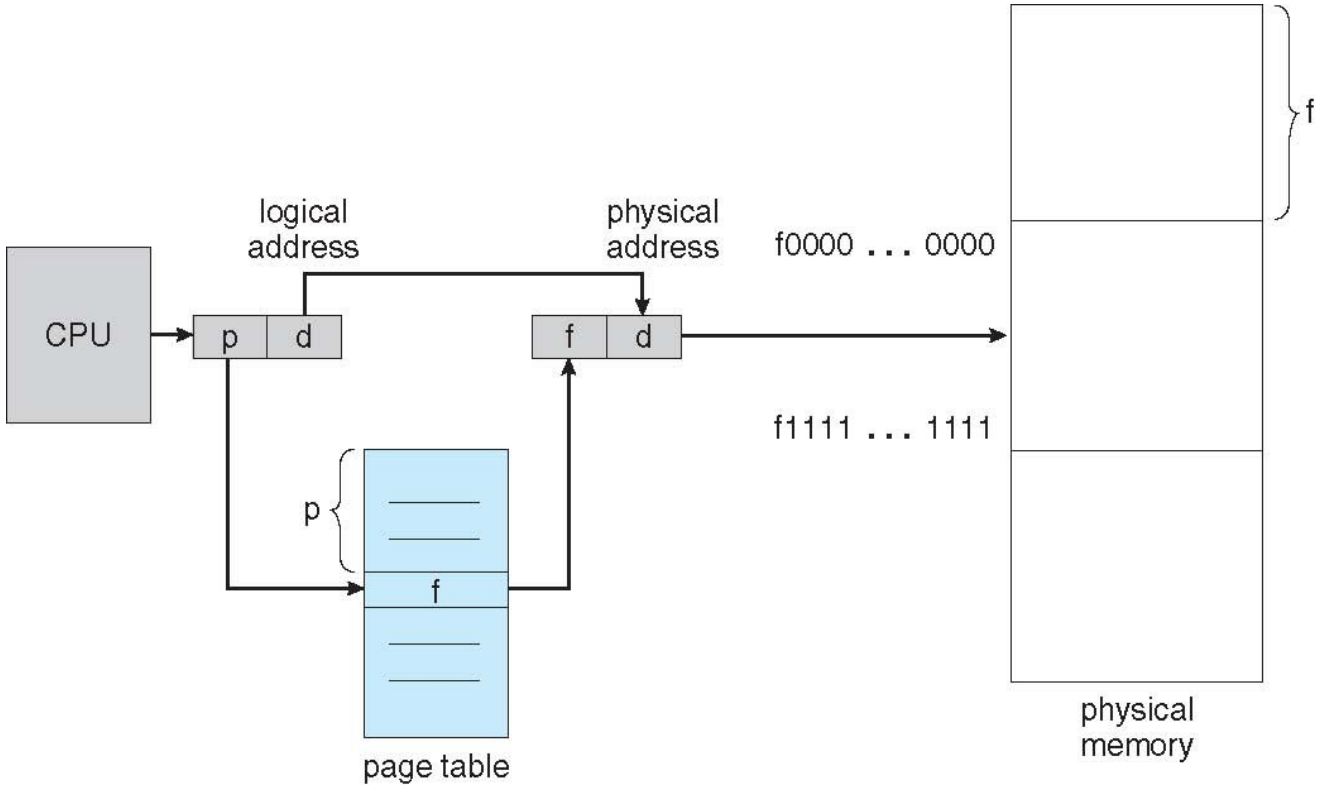
Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number** (*p*) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (*d*) – combined with base address to define the physical memory address that is sent to the memory unit

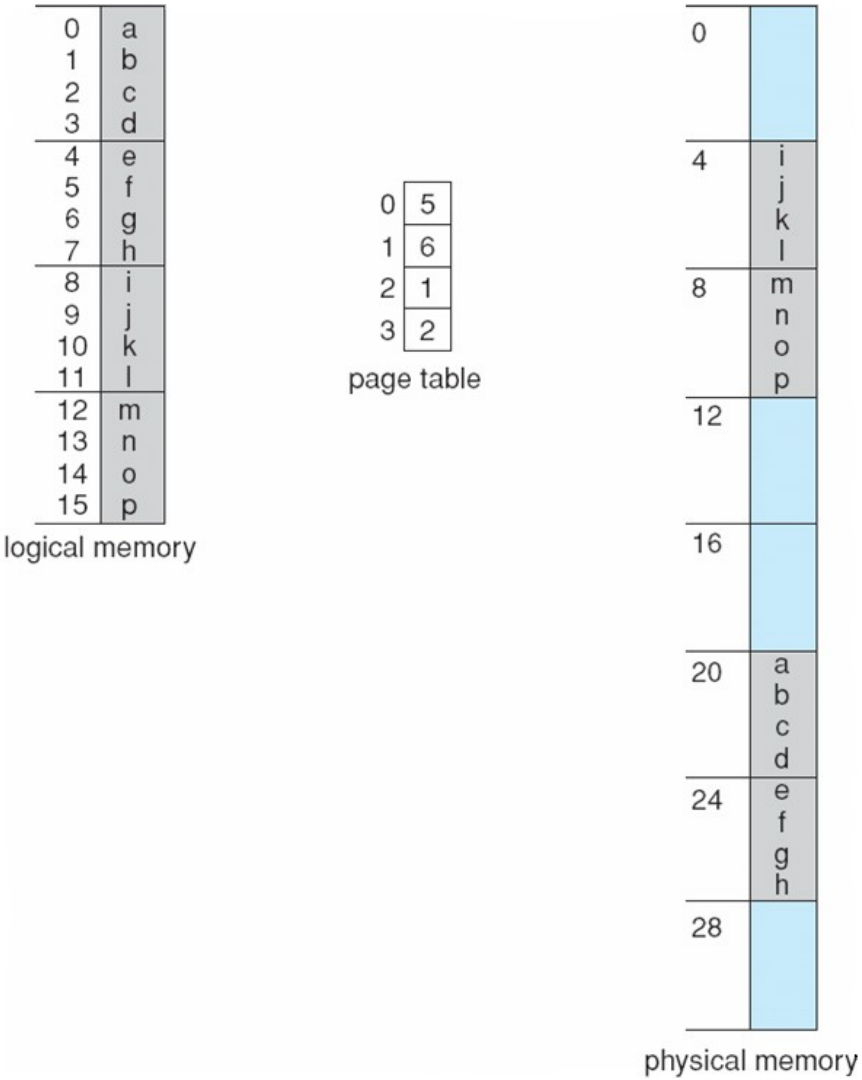


  - For given logical address space 2^m and page size 2^n

Paging Hardware



Paging Example



n=2 and *m*=4 32-byte memory and 4-byte pages

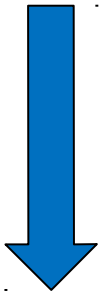
Paging: Example

- If the page size is equal to 1KB, in logical address 0000010111011110, how many pages are there in the logical memory space?

Paging: Example

- If the page size is equal to 1KB, in logical address 0000010111011110, how many bits are required for the page number?
  - 1 KB = 10 bits for page offset
  - 0000010111011110

16 bits



16-10=6 bits for the page #

So 2^6 pages in memory space

Paging: Example

- There are 4 pages in a logical address space and each page has 2 words. If these pages are assigned to a physical address space with 8 frames, how many bits are required for physical and logical addresses?

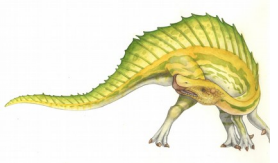
Paging: Example

- There are 4 pages in a logical address space and each page has 2 words. If these pages are assigned to a physical address space with 8 frames, how many bits are required for physical and logical addresses?
  - Logical address: 4 pages = 2bits, 2 words=1 bit => 3 bits
  - Physical address: 8 frames=3 bits, 2 words=1 bit => 4 bit

Paging (Cont.)

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - Worst case fragmentation = 1 frame – 1 byte
  - On average fragmentation = 1 / 2 frame size
- So small frame sizes desirable?
  - But each page table entry takes memory to track
  - Page sizes growing over time
    - Windows 10 supports page sizes of 4 KB and 2 MB .
    - Linux also supports two page sizes: a default page size (typically 4 KB) and an architecture-dependent larger page size called huge pages

getconf PAGESIZE



Paging (Cont.)

- Process view and physical memory now very different
  - The programmer views memory as one single space, containing only this one program
  - By implementation process can only access its own memory
- The frame table has one entry for each physical page frame, indicating whether the latter is free or allocated and, if it is allocated, to which page of which process (or processes).

Increasing or Decreasing page size

- What is the benefits of increasing page size?
- In what situation do you suggest to increase the page size?

Implementation of Page Table

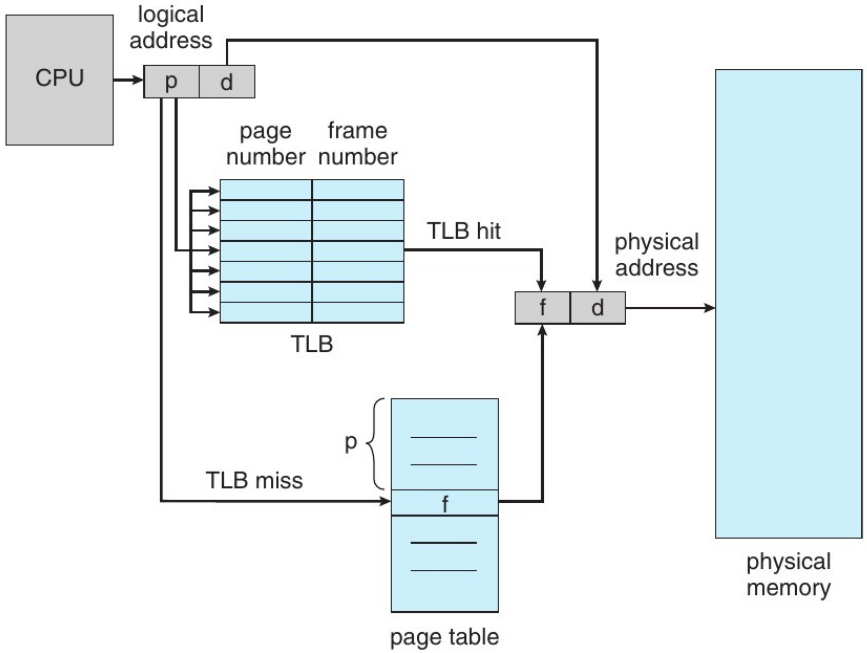
- Page table is kept in main memory
- **Page-table base register** (**PTBR**) points to the page table
- **Page-table length register** (**PTLR**) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses
  - One for the page table and one for the data / instruction
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers** (**TLBs**)

TLB

- The TLB is associative, high-speed memory.
- Each entry in the TLB consists of two parts: a key (or tag) and a value.
- When the associative memory is presented with an item, the item is compared with all keys simultaneously.
  - If the item is found, the corresponding value field is returned. The search is fast;
- a TLB lookup in modern hardware is part of the instruction pipeline, essentially adding no performance penalty.
- To be able to execute the search within a pipeline step, however, the TLB must be kept small

TLB

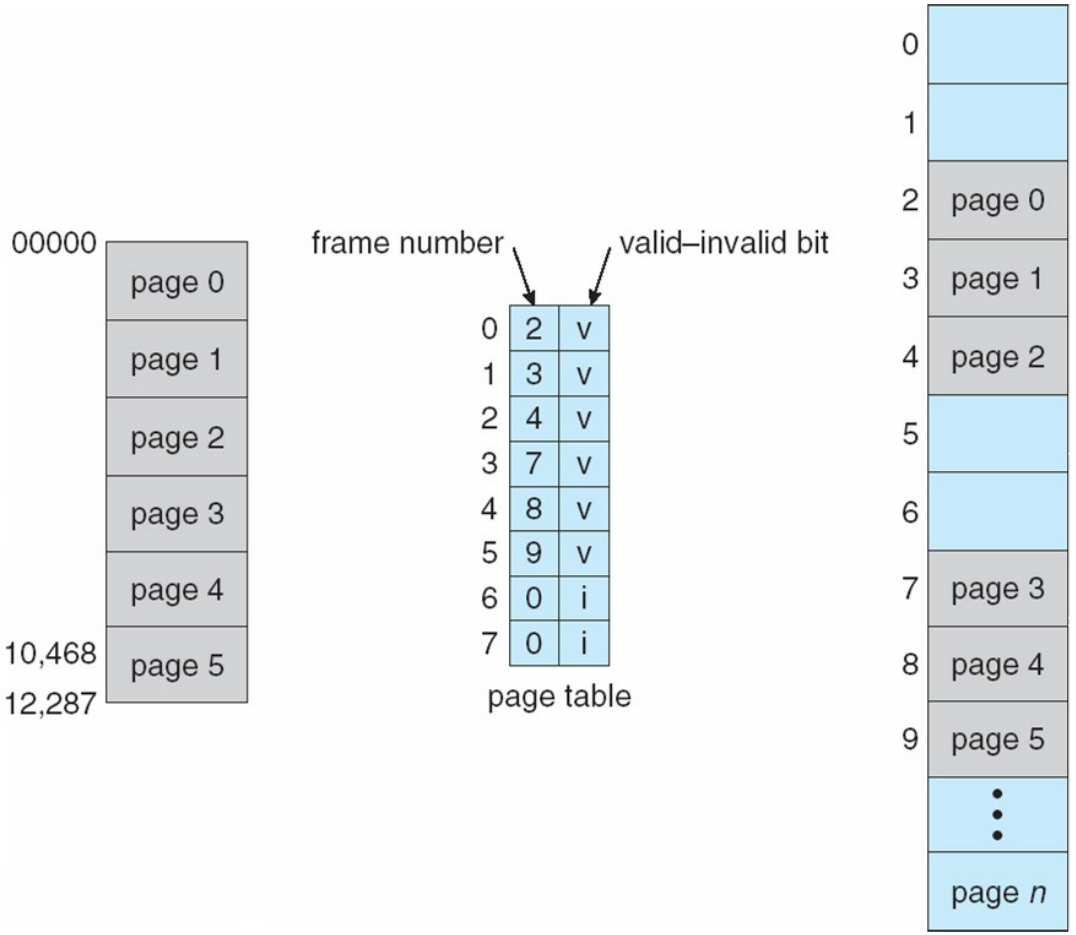
- The TLB contains only a few of the page-table entries.
- When a logical address is generated by the CPU, the MMU first checks if its page number is present in the TLB.
  - If the page number is found, its frame number is immediately available and is used
  - If the page number is not found, memory reference to the page table must be made.



Memory Protection

- Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed
  - Can also add more bits to indicate page execute-only, and so on
- **Valid-invalid** bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space
  - Or use **page-table length register** (**PTLR**)
- Any violations result in a trap to the kernel

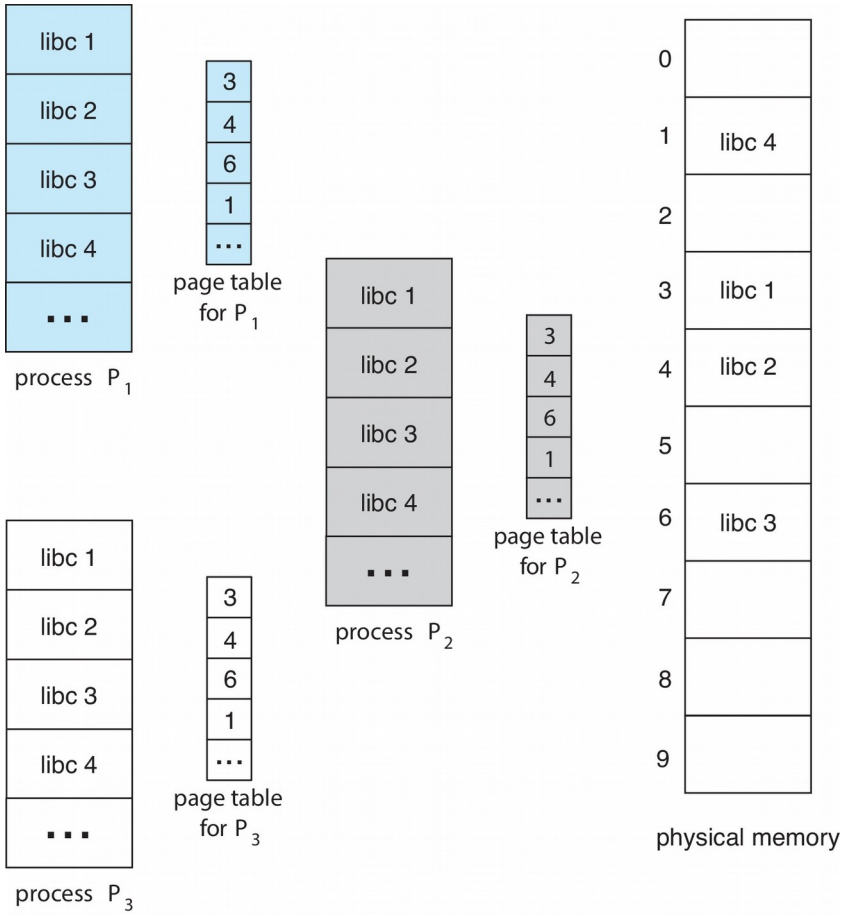
Valid (v) or Invalid (i) Bit In A Page Table



Shared Pages

- **Shared code**
  - One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example

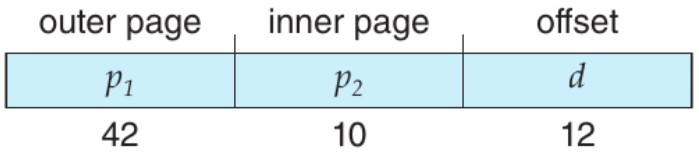


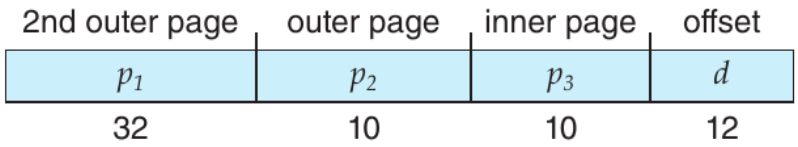
Structure of the Page Table

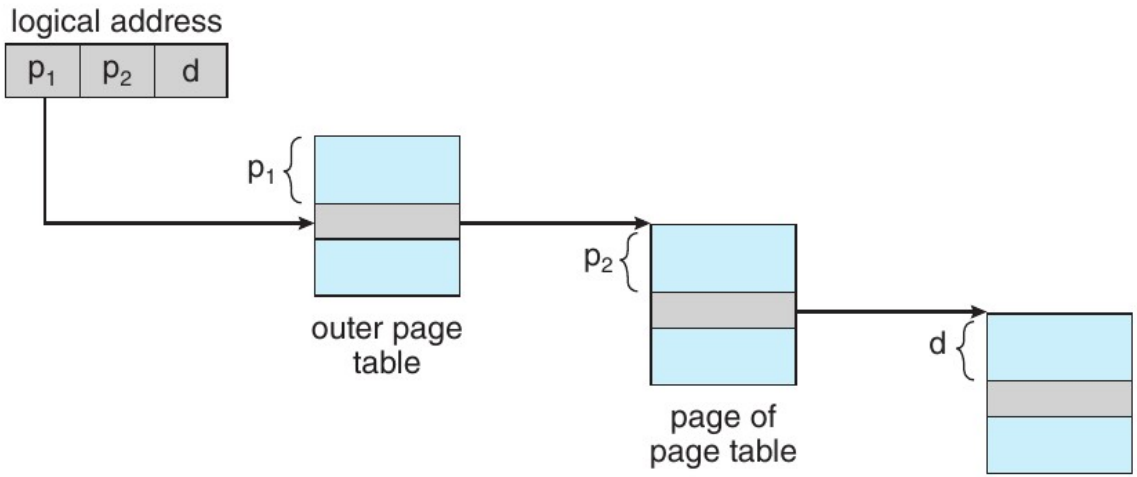
- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers
  - Page size of 4 KB (2^{12})
  - Page table would have 1 million entries ($2^{32} / 2^{12}$)
  - If each entry is 4 bytes → each process 4 MB of physical address space for the page table alone
    - Don't want to allocate that contiguously in main memory

Structure of the Page Table

- One simple solution is to divide the page table into smaller units
  - Hierarchical Paging
  - Hashed Page Tables
  - Inverted Page Tables







Paging and segmentation Combination

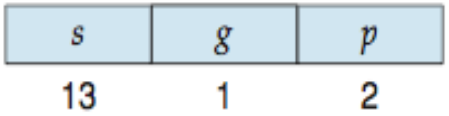
- What if paging and segmentation used simultaneously?
  - What is the fields of a logical address?
  - How the physical address is achieved from a logical address?
  - What are the benefits?

Example: The Intel IA-32 Architecture

- Supports both segmentation and segmentation with paging
  - Each segment can be 4 GB
  - Up to 16 K segments per process
  - Divided into two partitions
    - First partition of up to 8 K segments are private to process (kept in **local descriptor table** (**LDT**))
    - Second partition of up to 8K segments shared among all processes (kept in **global descriptor table** (**GDT**))

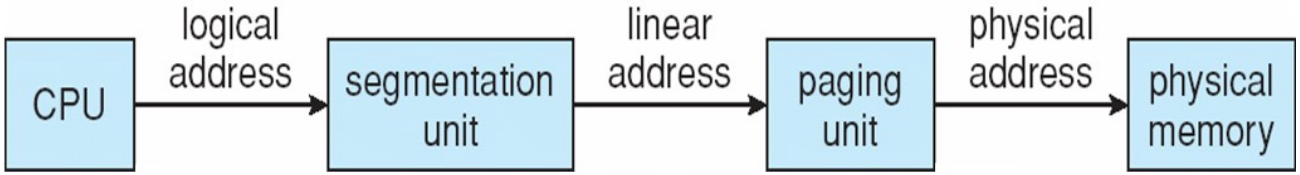
Example: The Intel IA-32 Architecture (Cont.)

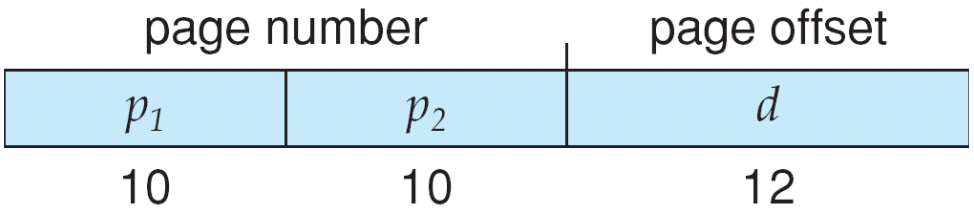
- CPU generates logical address
  - Selector given to segmentation unit
    - Which produces linear addresses



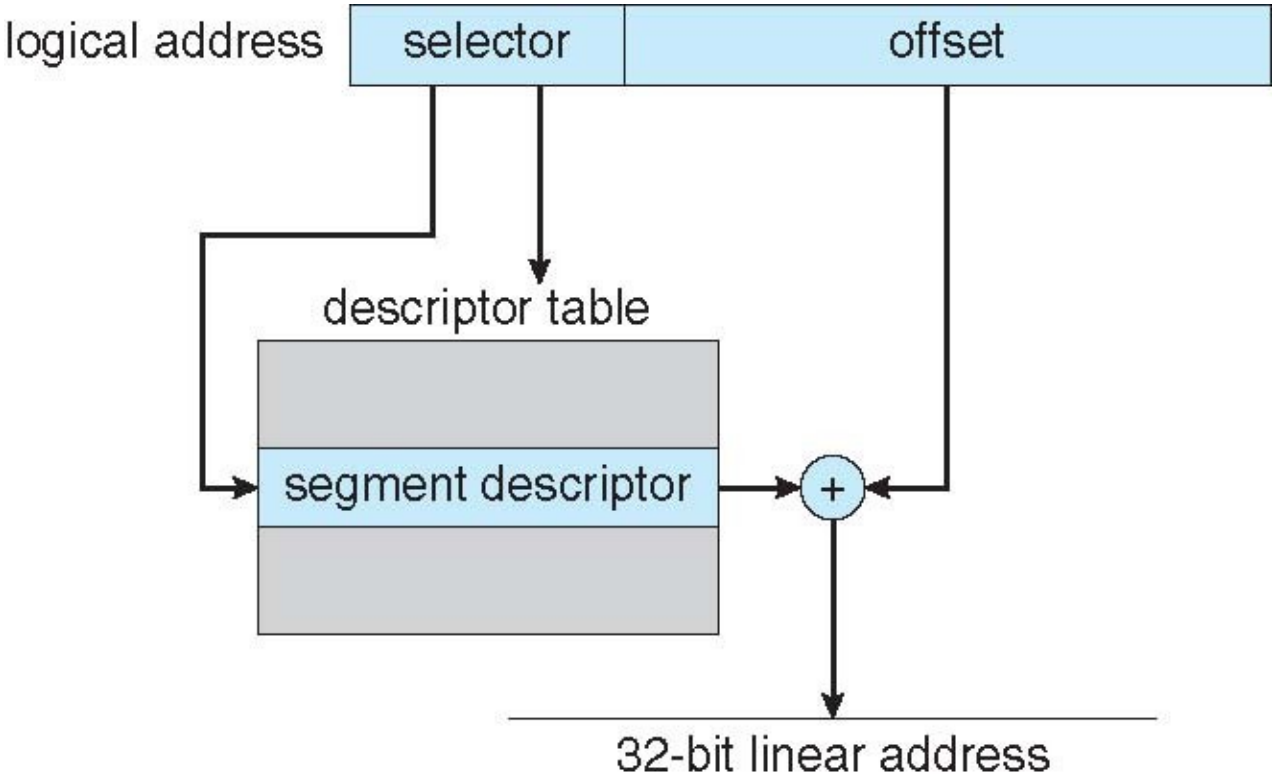
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU
    - Pages sizes can be 4 KB or 4 MB

Logical to Physical Address Translation in IA-32

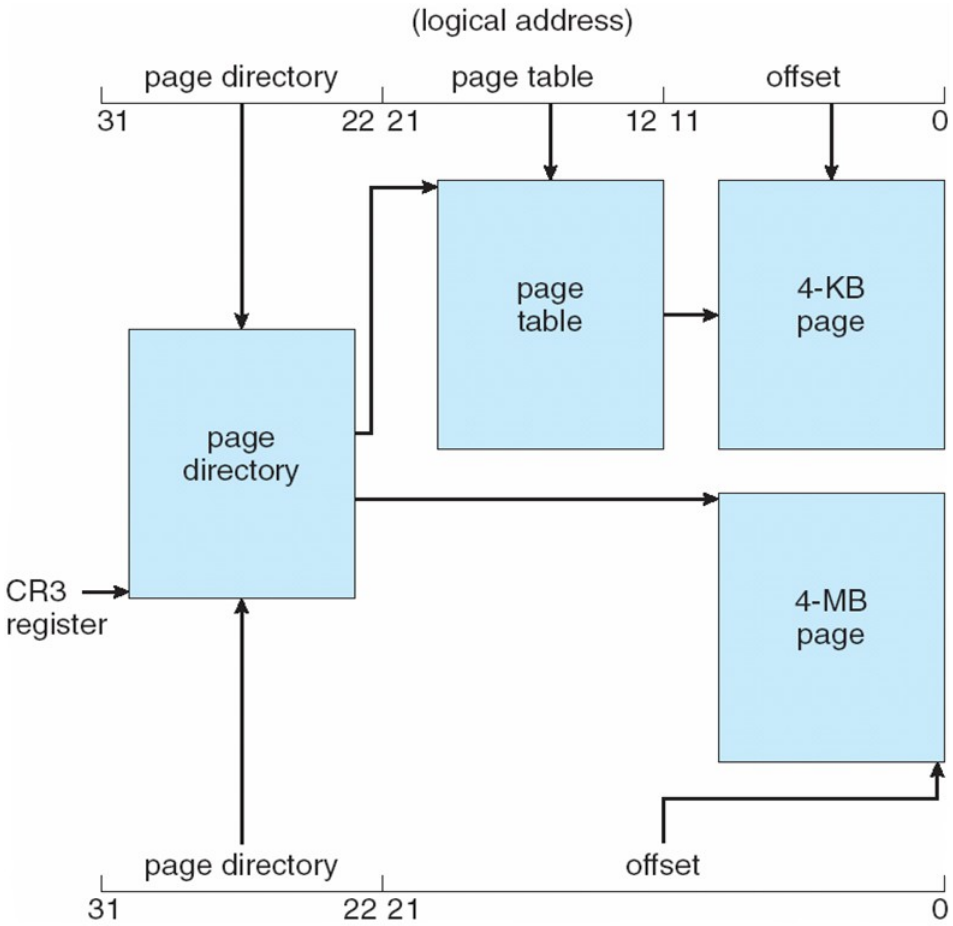




Intel IA-32 Segmentation

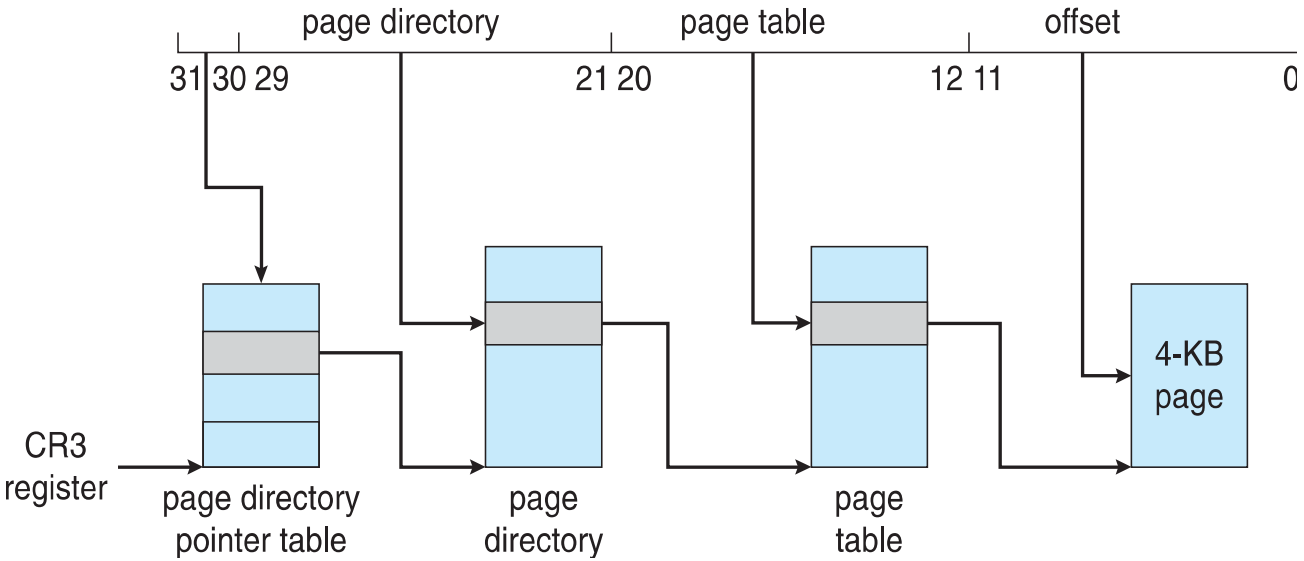


Intel IA-32 Paging Architecture



Intel IA-32 Page Address Extensions

- 32-bit address limits led Intel to create **page address extension** (**PAE**), allowing 32-bit apps access to more than 4GB of memory space
  - Paging went to a 3-level scheme
  - Top two bits refer to a **page directory pointer table**
  - Page-directory and page-table entries moved to 64-bits in size
  - Net effect is increasing address space to 36 bits – 64GB of physical memory



Intel x86-64

- Current generation Intel x86 architecture
- 64 bits is ginormous (> 16 exabytes)
- In practice only implement 48 bit addressing
  - Page sizes of 4 KB, 2 MB, 1 GB
  - Four levels of paging hierarchy
- Can also use PAE so virtual addresses are 48 bits and physical addresses are 52 bits