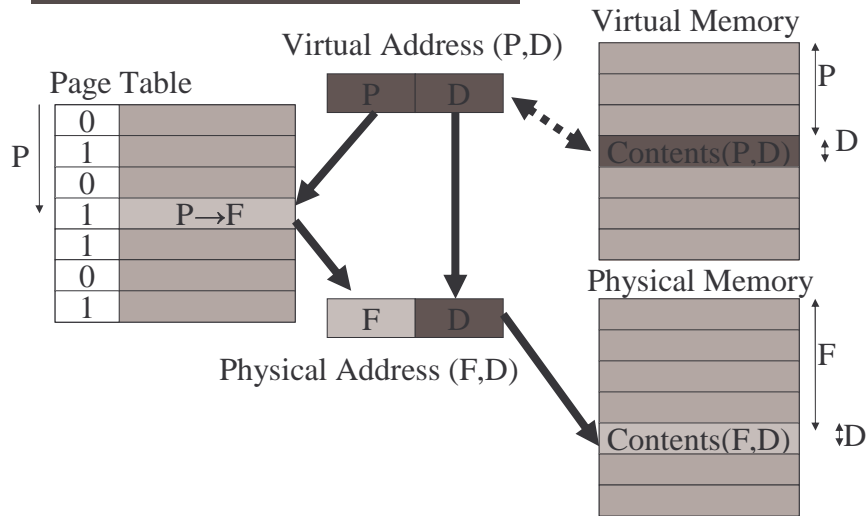


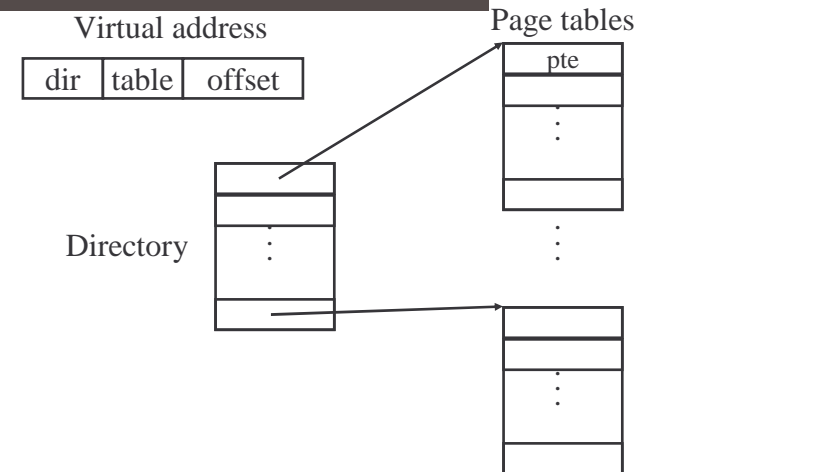
Page Mapping Hardware



Multilevel Page Tables

- Since the page table can be very large, one solution is to page the page table
- Divide the page number into
 - An index into a page table of second level page tables
 - A page within a second level page table
- Advantage
 - No need to keeping all the page tables in memory all the time
 - Only recently accessed memory's mapping need to be kept in memory, the rest can be fetched on demand

Multilevel Page Tables



What does this buy us? Sparse address spaces and easier paging

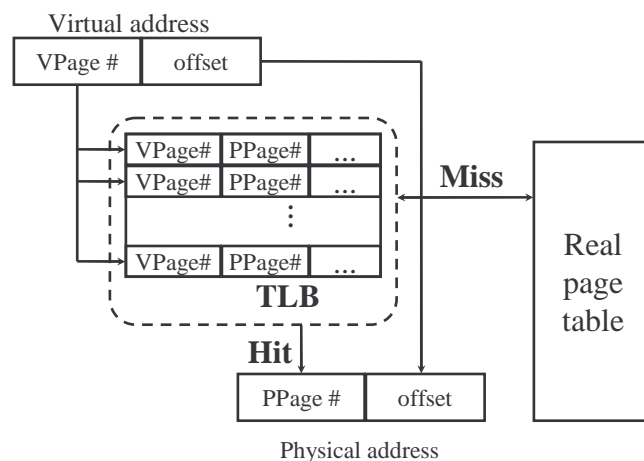
Page Fault

- Access a virtual page that is not mapped into any physical page
 - A fault is triggered by hardware
- Page fault handler (in OS's VM subsystem)
 - Find if there is any free physical page available
 - If no, evict some resident page to disk (swapping space)
 - Allocate a free physical page
 - Load the faulted virtual page to the prepared physical page
 - Modify the page table

Virtual-to-Physical Lookups

- Programs only know virtual addresses
 - The page table can be extremely large
- Each virtual address must be translated
 - May involve walking hierarchical page table
 - Page table stored in memory
 - So, each program memory reference requires several actual memory accesses
- Page table access has temporal locality
- Solution: cache "active" part of page table
 - TLB is an "associative memory"

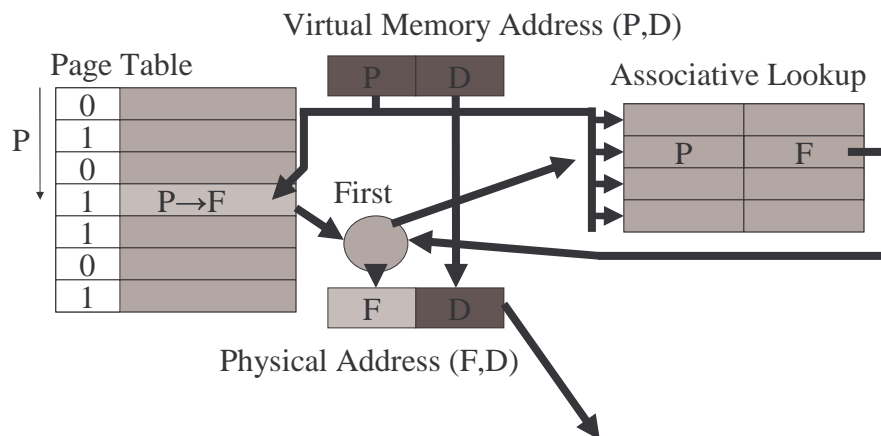
Translation Lookaside Buffer (TLB)



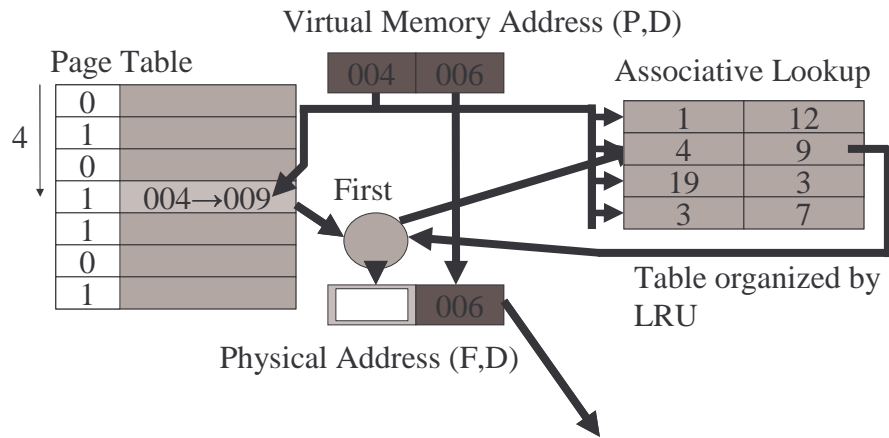
TLB Function

- If a virtual address is presented to MMU, the hardware checks TLB by comparing all entries simultaneously (in parallel).
- If valid match, the page is taken from TLB without going through page table.
- If not match
 - MMU detects miss and does an ordinary page table lookup.
 - It then evicts one page out of TLB and replaces it with the new entry, so that next time that page is found in TLB.

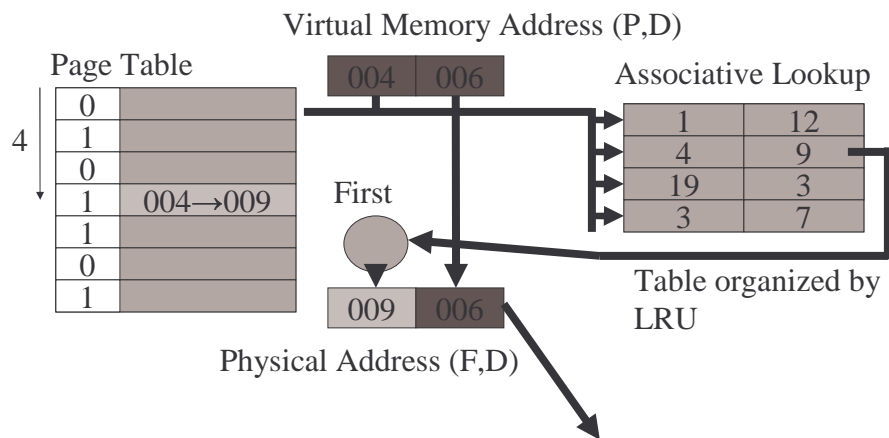
Page Mapping Hardware (Modified)



Page Mapping Example



Page Mapping Example (ref. again)



Bits in a TLB Entry

- Common (necessary) bits
 - Virtual page number: match with the virtual address
 - Physical page number: translated address
 - Valid
 - Protection bits: read, write, execution
- Optional (useful) bits
 - Process tag
 - Reference
 - Modify
 - Cacheable
- Include part of PTE
 - Example: x86

Implementation Issues

- TLB can be implemented using
 - Associative registers
 - Content-addressable (look-aside) memory
- TLB hit ratio (Page address cache hit ratio)
 - Percentage of time page translation found in associative memory

Hardware-Controlled TLB

- On a TLB miss (different from page fault)
 - Hardware walks through the page tables, generates a fault if the page containing the PTE is not present
 - VM software performs fault handling
 - Hardware loads the PTE into the TLB
 - Need to write back if there is no free entry
 - Restart the faulting instruction if needed
- On a TLB hit, hardware checks the protection bits
 - If no violation, pointer to page frame in memory
 - If violated, the hardware generates a protection fault
- Examples: IA-32, IA-64

Software-Controlled TLB

- On a miss in TLB, a "TLB miss" exception raised, VM software
 - Checks if the page containing the PTE is in memory
 - If no, performs page fault handling
 - Loads the PTE into the TLB
 - Writes back if there is no free entry
 - Restart the faulting instruction if needed
- On a hit in TLB, the hardware checks protection bits
 - If no violation, pointer to page frame in memory
 - If violated, the hardware generates a protection fault
- Example: SPARC, MIPS, Alpha, HP-PA, PowerPC

Issues

- Which TLB entry to be replaced?
 - Random
 - Pseudo Least Recently Used (LRU)
- What happens on a context switch?
 - Process tag: change TLB registers and process register
 - No process tag: Invalidate the entire TLB contents
- What happens when changing a page table entry?
 - Change the entry in memory
 - Invalidate the TLB entry