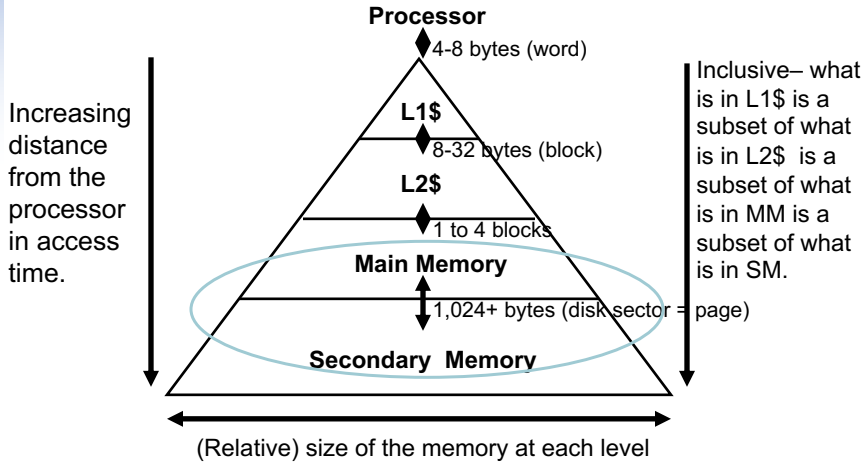
COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface
5th Edition

Chapter 5

Virtual Memory

Review: The Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as possible at the fastest speed and cheapest price.



How is the Hierarchy Managed?

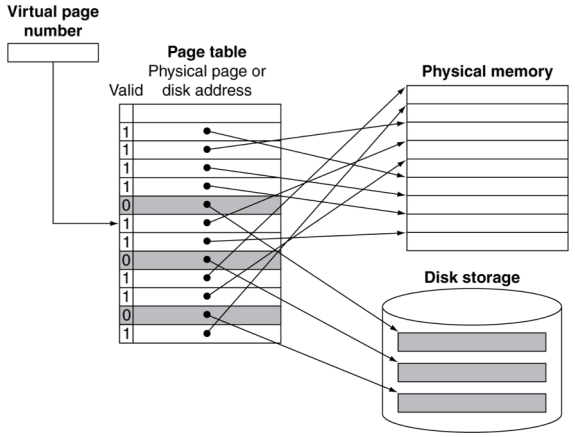
- Registers ↔ cache
  - ***By compiler or programmer.***
- Cache ↔ main memory
  - ***By the cache controller hardware.***
- Main memory ↔ disks
  - ***By the operating system*** (virtual memory).
  - Virtual to physical address mapping assisted by the hardware.

Virtual Memory

- Use main memory as a "cache" for secondary memory:
  - Allows efficient and ***safe*** sharing of memory among multiple programs.
  - Provides the ability to run programs larger than the size of physical memory.
  - Simplifies loading a program for execution by providing for code relocation (i.e., the code can be loaded anywhere in main memory).
- Each program is compiled into its own address space – a "virtual" address space:
  - During run-time, each ***virtual*** address must be translated to a ***physical*** address - an address in main memory.
  - Virtual Memory "block" is called a ***page.***
  - Virtual Memory translation "miss" is called a ***page fault.***

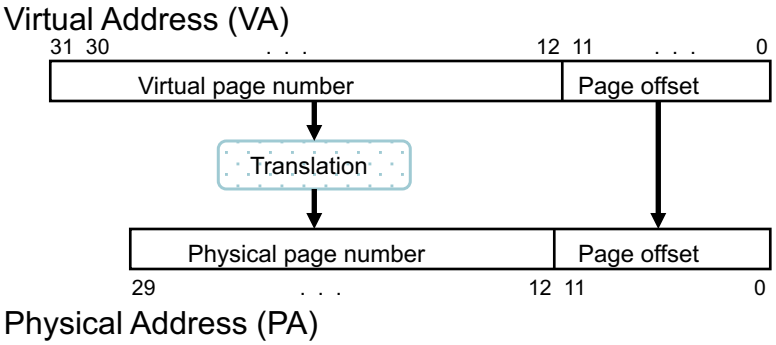
Two Programs Sharing Physical Memory

- A program's address space is divided into pages - fixed size - or segments - variable sizes:
  - The starting location of each page (either in main memory or in secondary memory) is contained in the program's ***page table.***



Address Translation

- A virtual address is translated to a physical address by a combination of hardware and software.
- Each memory request *first* requires an address ***translation*** from the virtual space to the physical space.



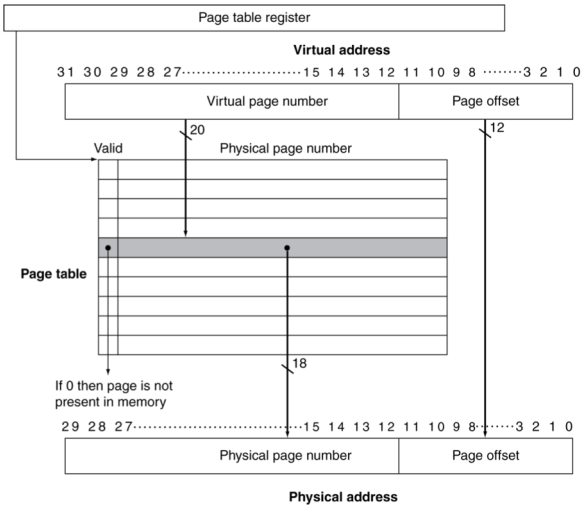
Page Fault Penalty

- On page fault, the entire page must be fetched from disk:
  - Takes millions of clock cycles.
  - Handled by the Operating System.
- Try to minimize page fault rate:
  - Fully associative placement of page in main memory.
  - Smarter replacement algorithms.

Page Tables

- Stores placement information:
  - A **Page Table** is an array of page table entries, indexed by virtual page number.
  - Page table register points to page in physical memory.
- If page is present in memory:
  - PTE stores the physical page number.
  - Plus other status bits (referenced, dirty, …).
- If page is not present:
  - **Page Fault** – OS gets involved.

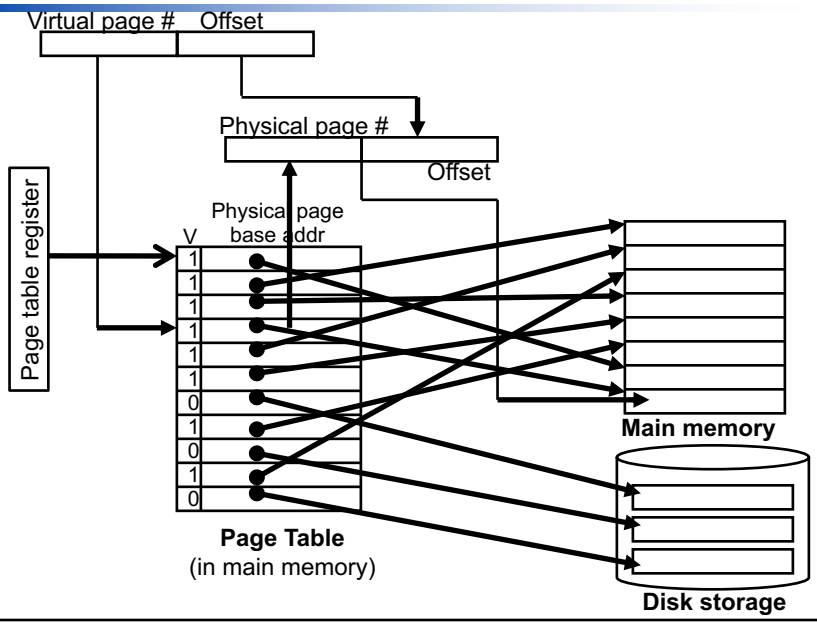
Translation Using a Page Table



Replacement and Writes

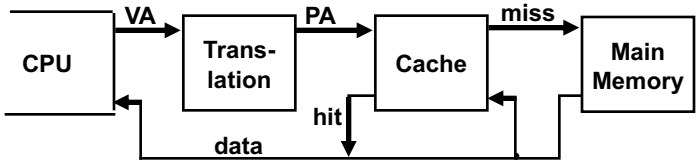
- To reduce page fault rate, prefer least-recently used (LRU) replacement:
  - Reference bit in Page-table-entry set to 1 on access to page.
  - Periodically cleared to 0 by OS.
  - A page with reference bit = 0 has not been used recently.
- Disk writes take millions of cycles:
  - Write-through is impractical so write-back is used.

Address Translation Summary



Virtual Addressing with a Cache

- It takes an ***extra*** memory access to translate a Virtual Address to a Physical Address via the Page Table.

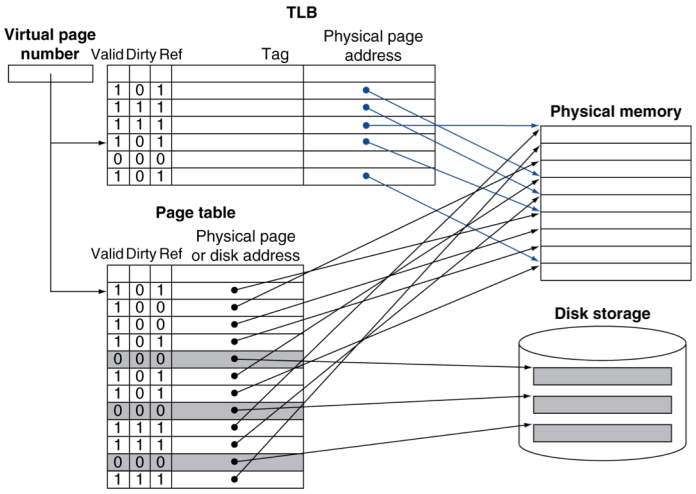


- This makes cache accesses very expensive (if every access was really *two* accesses).
- The hardware fix is to use a ***Translation Lookaside Buffer (TLB)*** – a small cache that keeps track of recently used address mappings to avoid having to do a page table lookup.

Fast Translation Using a TLB

- TLB's work well because access to page tables has good locality:
  - Use a fast cache of Page-Table-Entries within the CPU.
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate.
  - Misses can be handled by hardware or software.
- Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped.

Fast Translation Using a TLB



A TLB in the Memory Hierarchy

- A TLB miss – is it a page fault or merely a TLB miss?
  - If the page is loaded into main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB:
    - Takes 10's of cycles to find and load the translation info into the TLB.
  - If the page is not in main memory, then it's a true page fault:
    - Takes 1,000,000's of cycles to service a page fault.
- TLB misses are much more frequent than true page faults.

TLB Event Combinations

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – this is what we want!
Hit	Hit	Miss	Yes – although the page table is not checked if the TLB hits (Page fault).
Miss	Hit	Hit	Yes – TLB miss, PA in page table.
Miss	Hit	Miss	Yes – TLB miss, PA in page table, but data not in cache (Page fault).
Miss	Miss	Miss	Yes – page fault (OS allocates new PT entry).
Hit	Miss	Miss/ Hit	Impossible – TLB cannot Hit if Page Table misses.
Miss	Miss	Hit	Impossible – data not allowed in cache if No Page Table entry.

Memory Protection

- Different tasks can share parts of their virtual address spaces:
  - But need to protect against errant access.
  - Requires OS assistance.
- Hardware support for OS protection:
  - Privileged supervisor mode (aka kernel mode).
  - Privileged instructions.
  - Page tables and other state information only accessible in supervisor mode.

Some Virtual Memory Design Parameters

	VM Page	TLBs
Total size	16,000 to 250,000 words	16 to 512 entries
Total size (KB)	250,000 to 1,000,000,000	0.25 to 16
Block size (B)	4000 to 64,000	4 to 8
Hit time		0.5 to 1 clock cycle
Miss penalty (clocks)	10,000,000 to 100,000,000	10 to 100
Miss rates	0.00001% to 0.0001%	0.01% to 1%

2-Level TLB Organization

Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

Two Machines' TLB Parameters

	Intel Nehalem	AMD Barcelona
Address sizes	48 bits (vir); 44 bits (phy)	48 bits (vir); 48 bits (phy)
Page size	4KB	4KB
TLB organization	L1 TLB for instructions and L1 TLB for data per core; both are 4-way set assoc.; LRU L1 ITLB has 128 entries, L2 DTLB has 64 entries L2 TLB (unified) is 4-way set assoc.; LRU L2 TLB has 512 entries TLB misses handled in hardware	L1 TLB for instructions and L1 TLB for data per core; both are fully assoc.; LRU L1 ITLB and DTLB each have 48 entries L2 TLB for instructions and L2 TLB for data per core; each are 4-way set assoc.; round robin LRU Both L2 TLBs have 512 entries TLB misses handled in hardware

Two Machines' TLB Parameters

	Intel P4	AMD Opteron
TLB organization	1 TLB for instructions and 1TLB for data Both 4-way set associative Both use ~LRU replacement Both have 128 entries TLB misses handled in hardware	2 TLBs for instructions and 2 TLBs for data Both L1 TLBs fully associative with ~LRU replacement Both L2 TLBs are 4-way set associative with round-robin LRU Both L1 TLBs have 40 entries Both L2 TLBs have 512 entries TLB misses handled in hardware

The Hardware/Software Boundary

- What parts of the virtual to physical address translation are done by or assisted by the hardware?
  - Translation Lookaside Buffer (TLB) that caches the recent translations:
    - TLB access time is part of the cache hit time.
    - May allot an extra stage in the pipeline for TLB access.
  - Page table storage, fault detection, and updating:
    - Page faults result in precise interrupts that are then handled by the OS.
    - Hardware must support Dirty and Reference bits in the Page Tables.

Summary: Questions for the Memory Hierarchy

- Q1: Where can an entry be placed in the cache?
  (Entry placement)
- Q2: How is an entry found if it is in the cache?
  (Entry identification)
- Q3: Which entry should be replaced on a miss?
  (Entry replacement)
- Q4: What happens on a write?
  (Write strategy)

Q1&Q2: Where can an entry be placed/found?

	# of sets	Entries per set
Direct mapped	# of entries	1
Set associative	(# of entries)/ associativity	Associativity (typically 2 to 16)
Fully associative	1	# of entries

	Location method	# of comparisons
Direct mapped	Index	1
Set associative	Index the set; compare set's tags	Degree of associativity
Fully associative	Compare all entries' tags	# of entries

Q3: Which entry should be replaced on a miss?

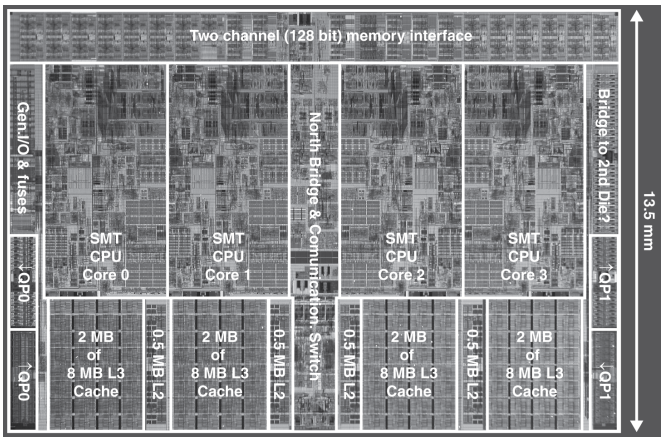
- Easy for direct mapped – only one choice.
- Set associative or fully associative:
  - Random.
  - LRU (Least Recently Used).
- For a 2-way set associative cache, random replacement has a miss rate about 1.1 times higher than LRU.
- LRU is too costly to implement for high levels of associativity (> 4-way) since tracking the usage information is costly.

Q4: What happens on a write?

- ***Write-through*** – The information is written to the entry in the current memory level *and* to the entry in the next level of the memory hierarchy:
  - Always combined with a write buffer so write-waits to next level memory can be eliminated (if the write buffer doesn't fill).
- ***Write-back*** – The information is written only to the entry in the current memory level. The modified entry is written to next level of memory only when it is replaced.
  - Need a dirty bit to keep track of whether the entry is clean or dirty.
  - Virtual memory systems always use write-back.

Multilevel On-Chip Caches

Intel Nehalem 4-core processor



Per core: 32KB L1 I-cache, 32KB L1 D-cache, 512KB L2 cache

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available

Summary

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time:
    - Temporal Locality - Locality in Time.
    - Spatial Locality - Locality in Space.
- Caches, TLBs, Virtual Memory all understood by examining how they deal with the four questions:
  1. Where can entry be placed?
  2. How is entry found?
  3. What entry is replaced on miss?
  4. How are writes handled?
- Page Tables map virtual address to physical address:
  - TLBs are important for fast translation.