CENG3420 Computer Organization & Design
Lecture 09: Virtual Memory Review

Bei Yu

Spring 2016

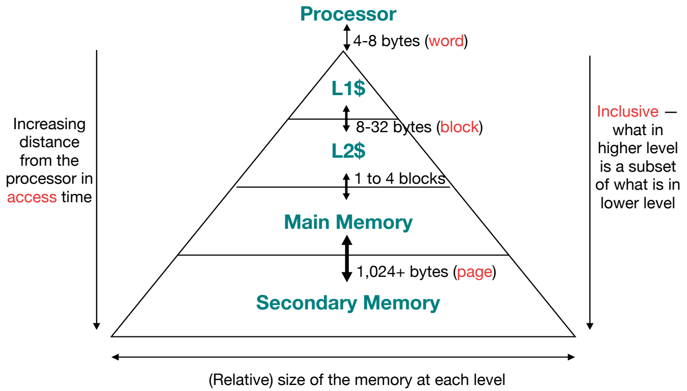
byu@cse.cuhk.edu.hk

香港中文大學
The Chinese University of Hong Kong

Review: Memory Hierarchy

Take advantage of principle of locality, present the user:

- as much memory as is available
- cheapest technology
- at the speed offered by the fastest technology


Processor
4-8 bytes (word)
L1$
8-32 bytes (block)
L2$
1 to 4 blocks
Main Memory
1,024+ bytes (page)
Secondary Memory
Increasing distance from the processor in access time
Inclusive — what in higher level is a subset of what is in lower level
(Relative) size of the memory at each level

Review: Reducing Cache Miss Rates #1

Direct mapped cache:

- a memory block maps to exactly one cache block

Fully associative cache:

- a memory block maps to any cache block

Review: Reducing Cache Miss Rates #1

Direct mapped cache:

- a memory block maps to exactly one cache block

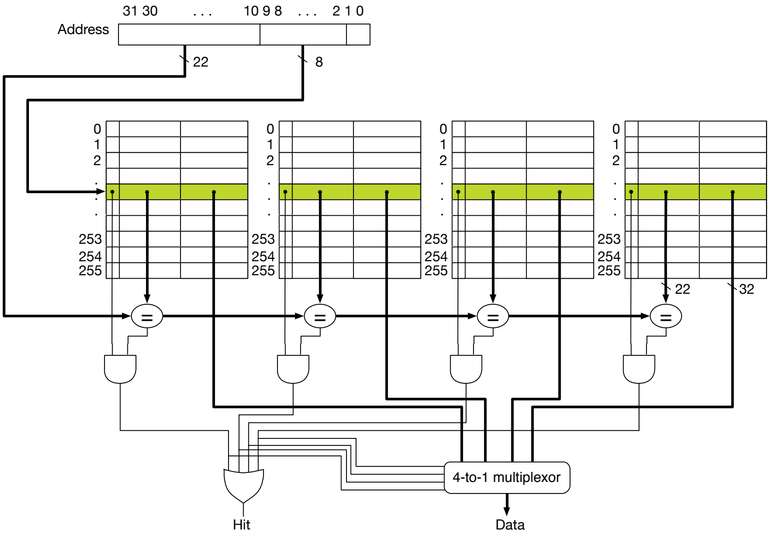
Fully associative cache:

- a memory block maps to any cache block

N-Way Set Associative Cache:

- A compromise is to divide the cache into sets
- `index` field maps a memory block to a unique set
- can be placed in any way of that set

Review: 4-Way Set Associative Cache



- $2^8 = 256$ sets each with four ways (each with one block)

Virtual Memory

- Use main memory as a “cache” for secondary memory
- Each program is compiled into its own virtual address space
- What makes it work? Principle of Locality

Virtual Memory

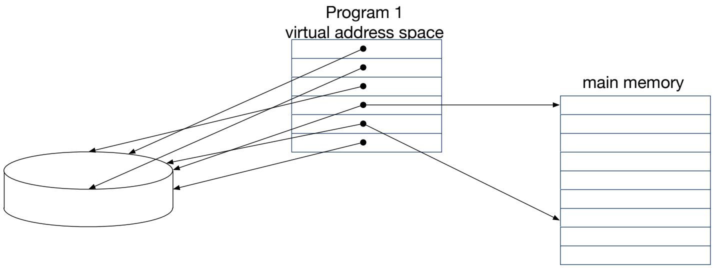
- Use main memory as a "cache" for secondary memory
- Each program is compiled into its own virtual address space
- What makes it work? Principle of Locality

Why virtual memory?

- During run-time, virtual address is translated to a physical address
- Efficient & safe sharing memory among multiple programs
- Ability to run programs larger than the size of physical memory
- Code relocation: code can be loaded anywhere in main memory

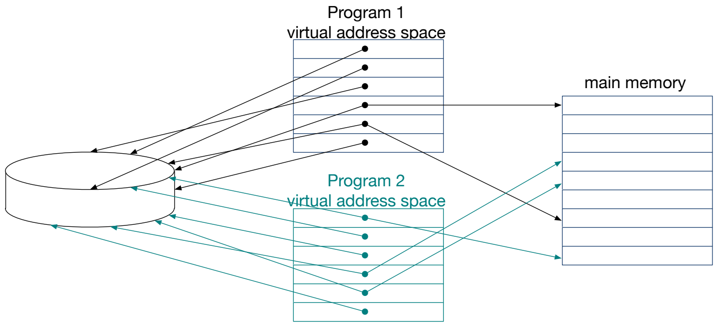
Two Programs Sharing Physical Memory

- A program's address space is divided into pages (fixed size) or segments (variable sizes)



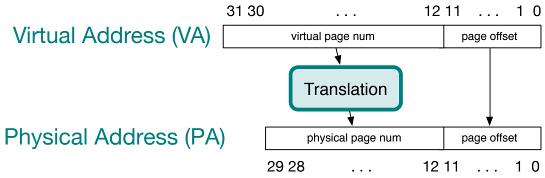
Two Programs Sharing Physical Memory

- A program's address space is divided into pages (fixed size) or segments (variable sizes)

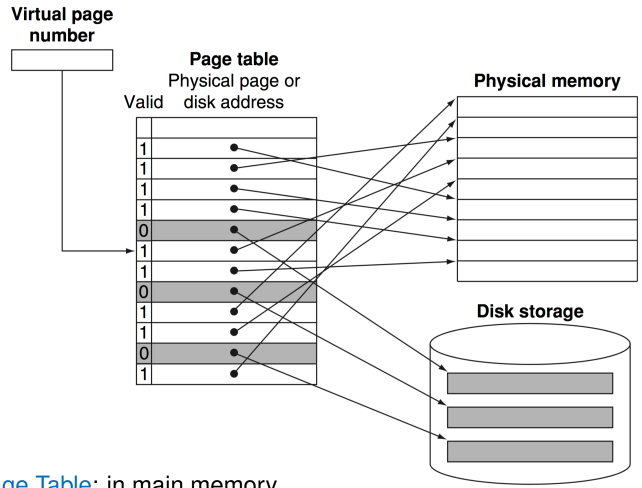


Address Translation

- Virtual address → physical address by combination of HW/SW
- Each memory request needs first an address translation
- Page Fault: a virtual memory miss



Address Translation Mechanisms

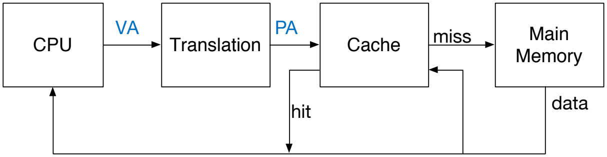


- Page Table: in main memory
- Process: page table + program counter + registers

Virtual Addressing with a Cache

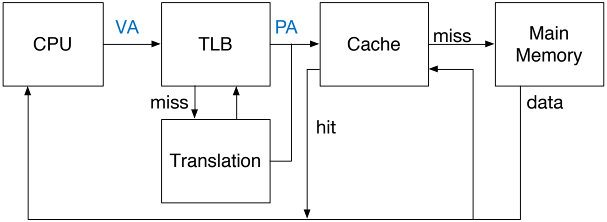
Disadvantage of virtual addressing:

- One extra memory access to translate a VA to a PA
- memory (cache) access very expensive...

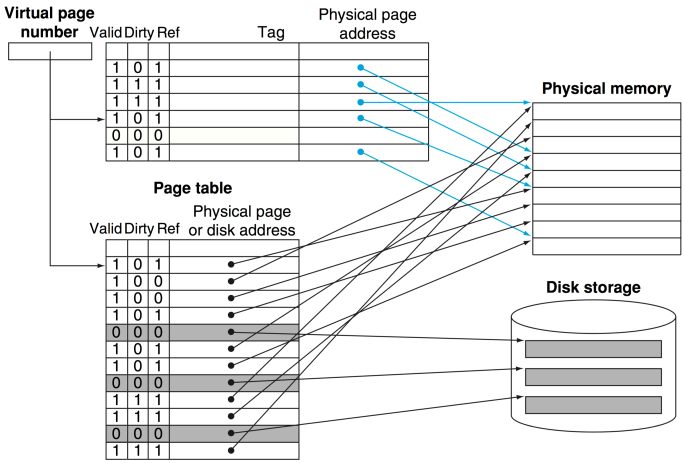


Translation Look-aside Buffer (TLB)

- A small cache: keeps track of recently used address mappings
- Avoid page table lookup



Translation Look-aside Buffer (TLB)



- Dirty bit:
- Ref bit:

More about TLB

Organization:

- Just like any other cache, can be fully associative, set associative, or direct mapped.

Access time:

- Faster than cache: due to smaller size
- Typically not more than 512 entries even on high end machines

A TLB miss:

- If the page is in main memory: miss can be handled; load translation info from page table to TLB
- If the page is NOT in main memory: page fault

TLB Event Combinations

- TLB / Cache miss: page / block not in “cache”
- Page Table miss: page NOT in memory

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	
Hit	Hit	Miss	
Miss	Hit	Hit	
Miss	Hit	Miss	
Miss	Miss	Miss	
Hit	Miss	Miss / Hit	
Miss	Miss	Hit	

TLB Event Combinations

- TLB / Cache miss: page / block not in "cache"
- Page Table miss: page NOT in memory

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although page table is not checked if TLB hits
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
Miss	Miss	Miss	Yes – page fault
Hit	Miss	Miss / Hit	
Miss	Miss	Hit	

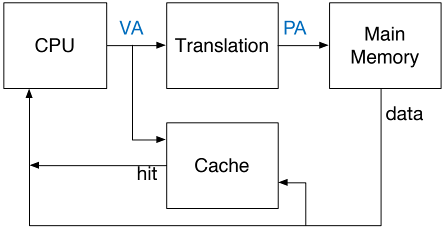
TLB Event Combinations

- TLB / Cache miss: page / block not in “cache”
- Page Table miss: page NOT in memory

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although page table is not checked if TLB hits
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
Miss	Miss	Miss	Yes – page fault
Hit	Miss	Miss / Hit	Impossible – TLB translation not possible if page is not in memory
Miss	Miss	Hit	Impossible – data not allowd in cache if page is not in memory

Question: Why Not a Virtually Addressed Cache?

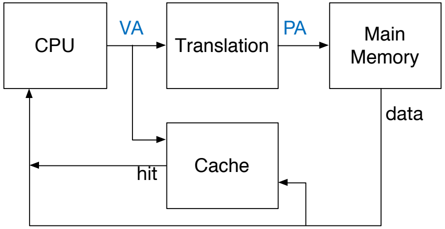
- Access Cache using virtual address (VA)
- Only address translation when cache misses



Answer:

Question: Why Not a Virtually Addressed Cache?

- Access Cache using virtual address (VA)
- Only address translation when cache misses



Answer:

- aliasing: 2 programs may share data w. different VAs for the same PA
- Coherence issues: must update all cache entries with same PAs

Q1: Where A Block Be Placed in Upper Level?

Scheme name	# of sets	Blocks per set
Direct mapped	# of blocks	1
Set associative	$\frac{\text{\# of blocks}}{\text{Associativity}}$	Associativity
Fully associative	1	# of blocks

Q1: Where A Block Be Placed in Upper Level?

Scheme name	# of sets	Blocks per set
Direct mapped	# of blocks	1
Set associative	$\frac{\text{\# of blocks}}{\text{Associativity}}$	Associativity
Fully associative	1	# of blocks

Q2: How Is Entry Be Found?

Scheme name	Location method	# of comparisons
Direct mapped	Index	1
Set associative	Index the set; compare set's tags	Degree of associativity
Fully associative	Compare all tags	# of blocks
	Separate page tables	0

Q3: Which Entry Should Be Replaced on a Miss?

- Direct mapped: only one choice
- Set associative or fully associative:
  - Random
  - LRU (Least Recently Used)

Note that:

- For a 2-way set associative, random replacement has a miss rate $1.1\times$ than LRU
- For high level associativity (4-way), LRU is too costly

Q4: What Happen On A Write?

- Write-Through:
  - The information is written in both the block in cache & the block in lower level of memory
  - Combined with write buffer, so write waits can be eliminated
  - ⊕:
  - ⊕:
- Write-Back:
  - The information is written only to the block in cache
  - The modification is written to lower level, only when the block is replaced
  - Need dirty bit: tracks whether the block is clean or not
  - Virtual memory always use write-back
  - ⊕:
  - ⊕:

Q4: What Happen On A Write?

- Write-Through:
  - The information is written in both the block in cache & the block in lower level of memory
  - Combined with write buffer, so write waits can be eliminated
  - ⊕: read misses don't result in writes
  - ⊕: easier to implement

- Write-Back:
  - The information is written only to the block in cache
  - The modification is written to lower level, only when the block is replaced
  - Need dirty bit: tracks whether the block is clean or not
  - Virtual memory always use write-back
  - ⊕:
  - ⊕:

Q4: What Happen On A Write?

- Write-Through:
  - The information is written in both the block in cache & the block in lower level of memory
  - Combined with write buffer, so write waits can be eliminated
  - $\oplus$: read misses don't result in writes
  - $\oplus$: easier to implement

- Write-Back:
  - The information is written only to the block in cache
  - The modification is written to lower level, only when the block is replaced
  - Need dirty bit: tracks whether the block is clean or not
  - Virtual memory always use write-back
  - $\oplus$: write with speed of cache
  - $\oplus$: repeated writes require only one write to lower level