CpE 313: Microprocessor Systems Design
Fall 2004

Credits:Asanovic @ MIT

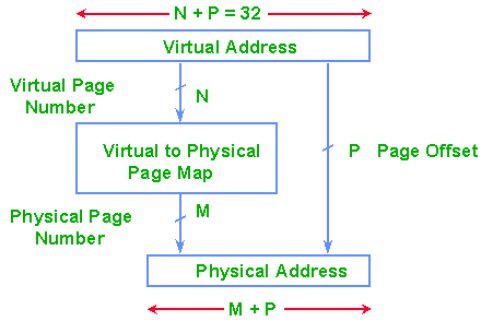
Handout 18
TLBs and Virtual Caches

November 23, 2004

Shoukat Ali

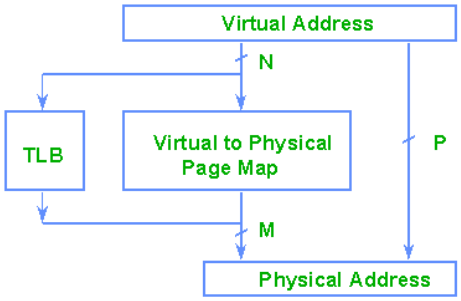
shoukat@umr.edu

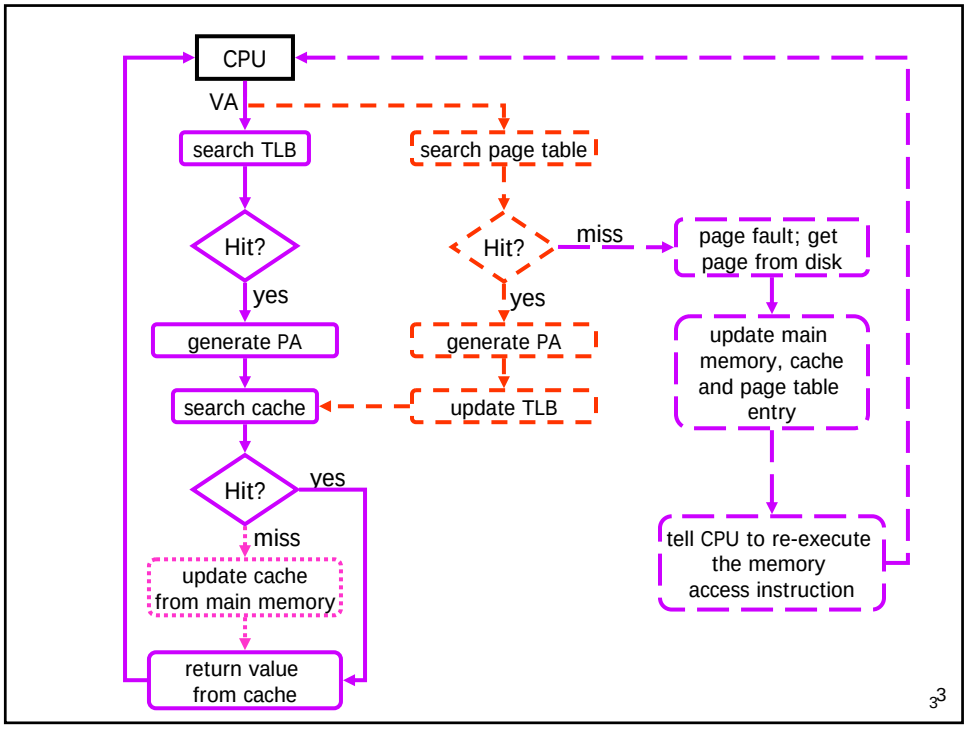
UMR UNIVERSITY OF MISSOURI-ROLLA
The Name. The Degree. The Difference.

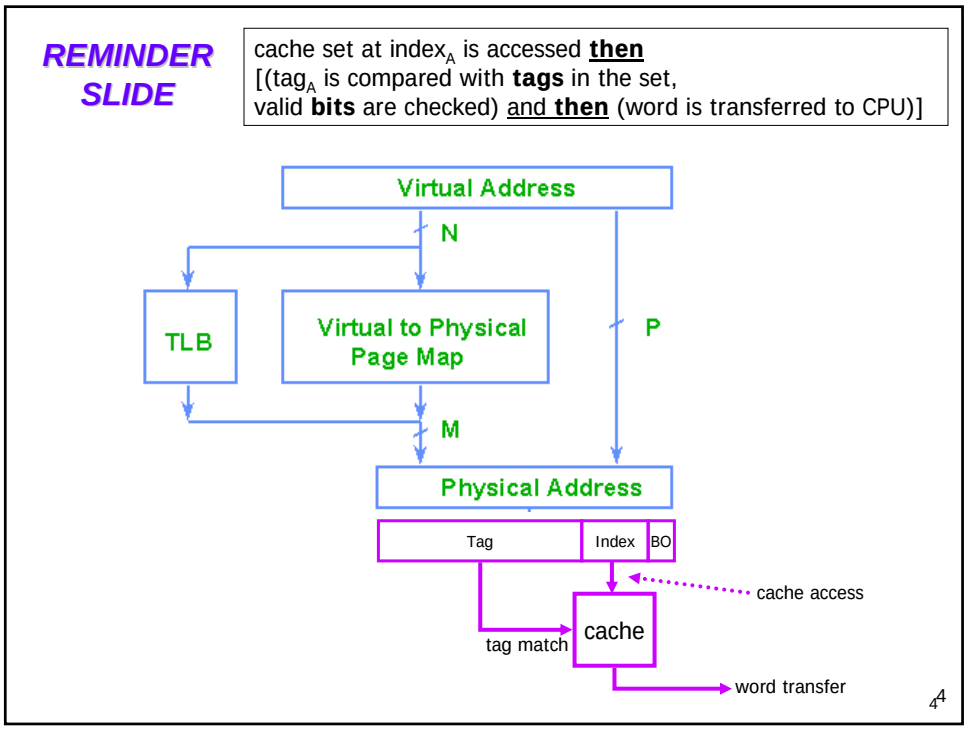


motivation: if the memory accesses have locality, then address translations must also have locality

A TLB is a small (usually) fully associative cache of recently used VPN → PPN mappings.

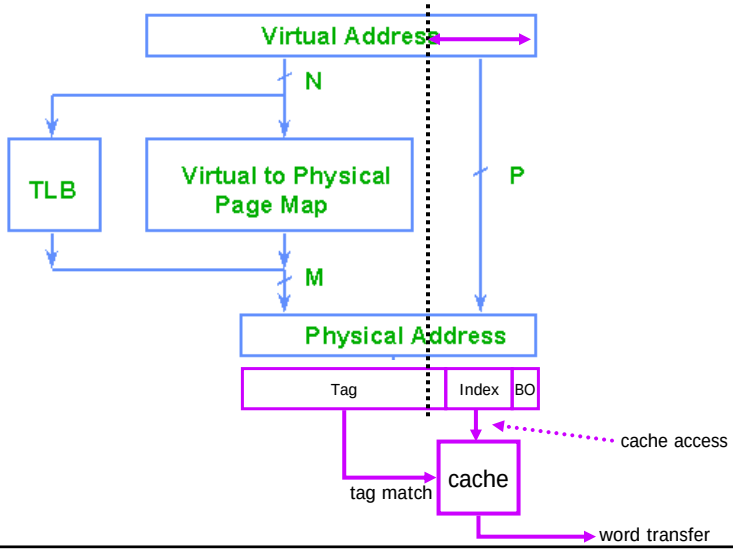



CPU
VA
search TLB
search page table
Hit?
Hit?
miss
page fault; get page from disk
yes
yes
generate PA
generate PA
update main memory, cache and page table entry
search cache
update TLB
Hit?
yes
miss
update cache from main memory
return value from cache
tell CPU to re-execute the memory access instruction
3


REMINDER SLIDE
cache set at index$_A$ is accessed then
[(tag$_A$ is compared with tags in the set,
valid bits are checked) and then (word is transferred to CPU)]
Virtual Address
N
TLB
Virtual to Physical Page Map
P
M
Physical Address
Tag
Index
BO
cache access
tag match
cache
word transfer
4

Great Observation: What If the Index + Block Offset Fall Inside Page Offset?

For cache access, it is not necessary to do virtual to physical address translation if index + block offset fall INSIDE the page offset.



5 5

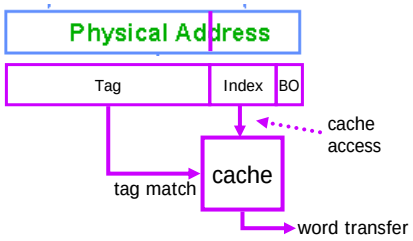
Virtually Indexed Caches

- a cache in which cache access phase uses the virtual address
- ***should everyone use virtually indexed caches?***

6 6

Consequences of Housing Index + BO in Page Offset

- assume we want (index + block offset) to fall inside page offset
  - don't want to do VA→ PA translation to access cache

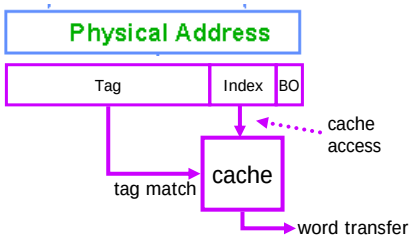


- this decision limits the cache size
  - recall: 2^{index} = cache size / (block size × associativity)
  - cache size = associativity * $2^{index+bo}$
  - if (index + bo) is already equal to page offset, then the only way to increase the cache size is by increasing associativity

7 7

Virtually Indexed, Physically Tagged Caches

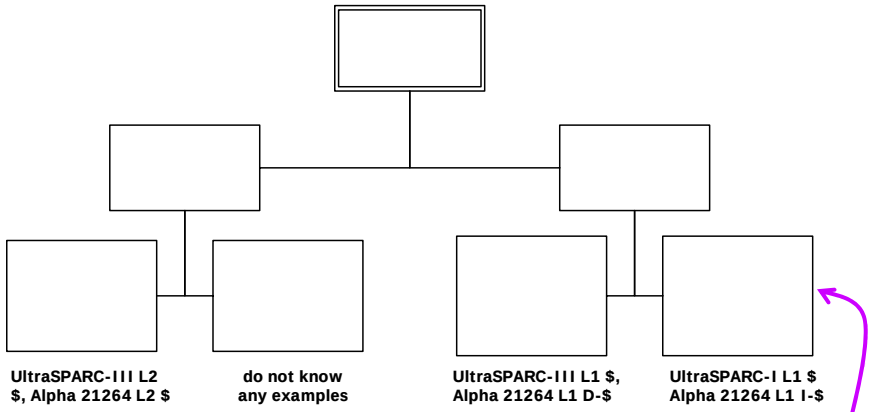
- what about tag check?



- still done with the tag taken from PA
- so our cache is really "virtually indexed, physically tagged"

8 8

Cache Landscape



9

An Example of a VI Cache

- recall: cache size = associativity * $2^{index+bo}$
- old question: why does IBM 3033 have such a high associativity cache when we think DMCs are "faster"

- the IBM 3033 system has 4KB pages
- it wants to access cache without doing the address translation first
- it wants a 64 KB cache
- what must be the associativity of such a cache?
  - maximum field length for index + bo = $\log_2$(page size) = 12
  - associativity = 64 KB / 4 KB = 16

- can you now see why IBM 3033 chose to have a large associativity cache?

10

Physically Indexed, Physically Tagged P/P

Another Example of a VI Cache

- recall: cache size = associativity * $2^{index+bo}$
- a system has 8KB pages
- it wants to access cache without doing the address translation first
- it wants a 16 KB cache

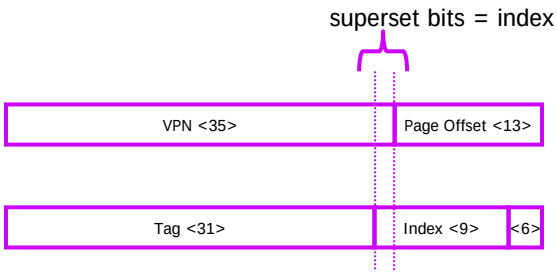
- what must be the associativity of such a cache?

- if the system above wanted a DMC, how big could it be?

New Rule

- if a VI cache wants to house index+ bo within the page offset, the cache size is limited to one page per way of associativity

What If A Part Of Index + Block Offset Falls Outside Page Offset?



For cache access, is it now necessary to do virtual to physical address translation?

The VI Instruction Cache in Alpha 21264

- Alpha 21264 has 8KB pages and a 2-way associative instruction cache ~~(very much like Pentium III)~~
- Alpha 21264 has a 64 KB cache ~~(unlike 16 KB in Pentium III)~~
- is it possible to accurately index the cache with index taken from the VA?
- how many superset bits does Alpha 21264 have?

Accessing and Searching the VI I-Cache in Alpha 21264

- Alpha 21264 needs 15 bits of index + bo
- it picks 13 bits from the VA, denote by v vvvv vvvv vvvv
- accesses 4 sets at cache indices
  00v vvvv vvvv vvvv, 01v vvvv vvvv vvvv,
  10v vvvv vvvv vvvv, 11v vvvv vvvv vvvv
  - these 4 sets are known as “superset”
- i.e., it accesses eight possible locations (2 for each set) in parallel for tags
- then uses the physical tag from TLB to do tag search

15