SE-292 High Performance Computing

Memory Hierarchy

R. Govindarajan
govind@serc

Reality Check

- Question 1: Are real caches built to work on virtual addresses or physical addresses?
- Question 2: What about multiple levels in caches?
- Question 3: Do modern processors use pipelining of the kind that we studied?

Virtual Memory System

- To support memory management when multiple processes are running concurrently.
  - Page based, segment based, ...
- Ensures protection across processes.
- Address Space: range of memory addresses a process can address (includes program (text), data, heap, and stack)
  - 32-bit address ⇒ 4 GB
  - with VM, address generated by the processor is virtual address

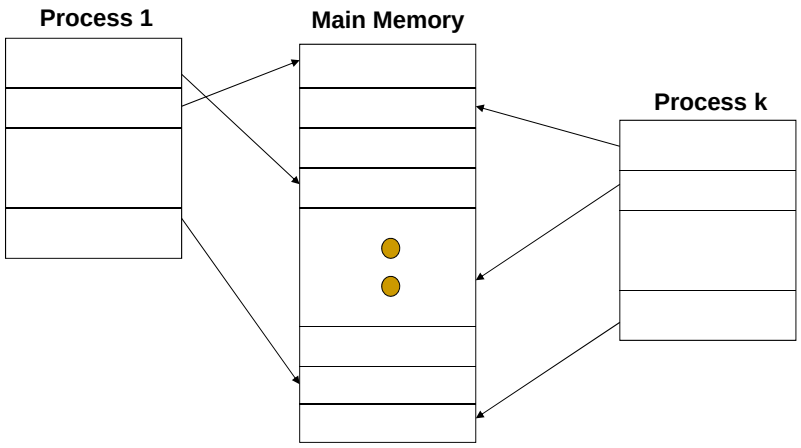
3

Page-Based Virtual Memory

- A process' address space is divided into a no. of pages (of fixed size).
- A page is the basic unit of transfer between secondary storage and main memory.
- Different processes share the physical memory
- Virtual address to be translated to physical address.

4

Virtual Pages to Physical Frame Mapping



5

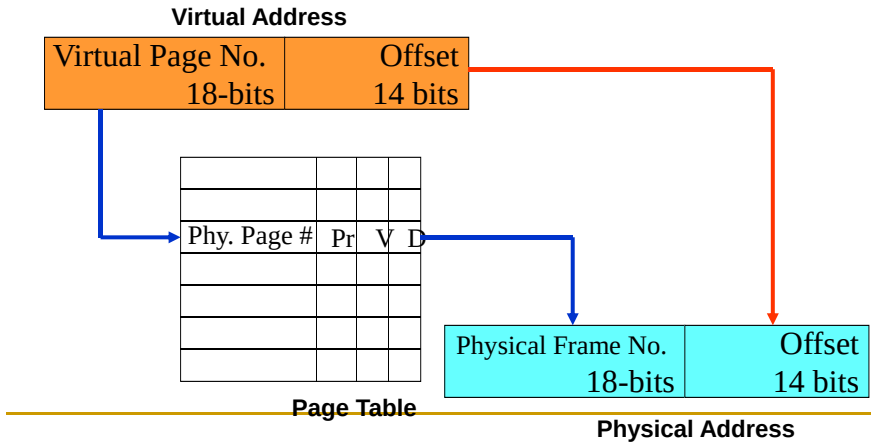
Page Mapping Info (Page Table)

- A page is mapped to any frame in the main memory.
- Where to store/access the mapping?
  - Page Table
  - Each process will have its own page table!
- Address Translation: virtual to physical address translation

6

Address Translation

- Virtual address (issued by processor) to Physical Address (used to access memory)



7

Memory Hierarchy: Secondary to Main Memory

- Analogous to Main memory to Cache.
- When the required virtual page is not in main memory: Page Hit
- When the required virtual page is not in main memory: Page fault
- Page fault penalty very high (~10's of mSecs.) as it involves disk (secondary storage) access.
- Page fault ratio shd. be very low (10^{-4} to 10^{-5})
- Page fault handled by OS.

8

Page Placement

- A virtual page is placed ***anywhere*** in physical memory (fully associative).
- Page Table keeps track of the page mapping.
- Separate page table for each process.
- Page table size is quite large!

  Assume 32-bit address space and 16KB page size.

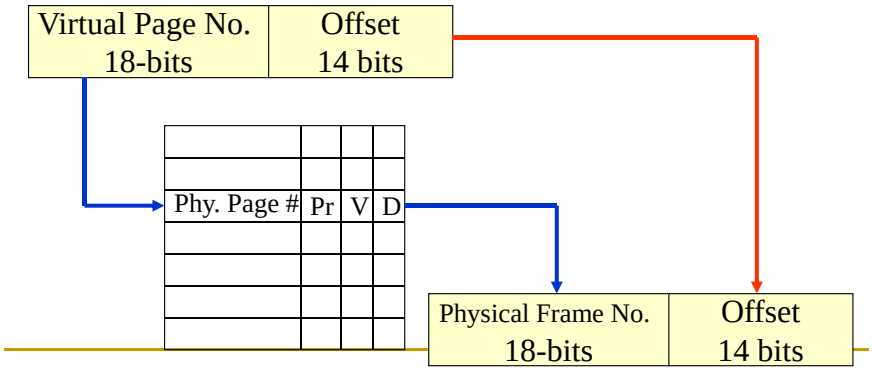
  # of entries in page table = $2^{32} / 2^{14} = 2^{18}$ = 256K

  Page table size = 256 K * 4B = 1 MB = 64 pages!
- Page table itself may be paged (multi-level page tables)!

9

Page Identification

- Use virtual page number to index into page table.
- Accessing page table causes one extra memory access!



10

Page Replacement

- Page replacement can use more sophisticated policies than in Cache.
  - Least Recently Used
  - Second-chance algorithm
  - Recency vs. frequency

Write Policies

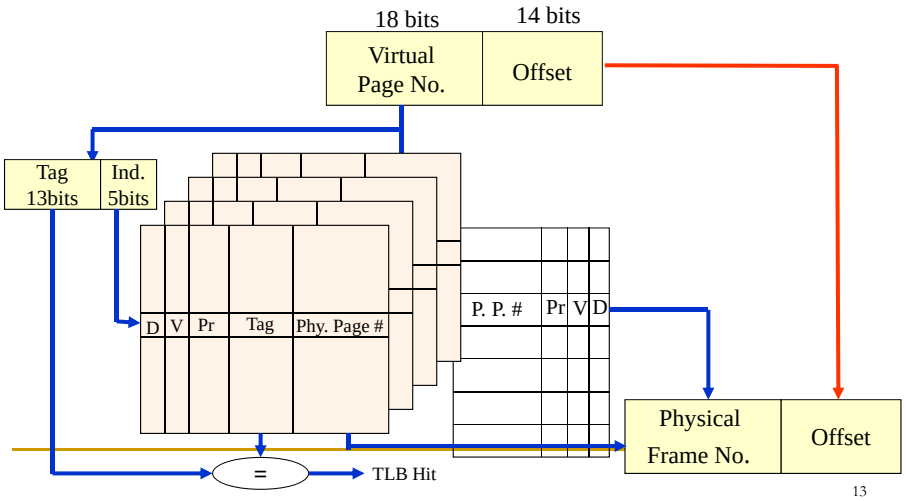
  - Write-back
  - Write-allocate

Translation Look-Aside Buffer

- Accessing the page tables causes one extra memory access!
- To reduce translation time, use translation look-aside buffer (TLB) which caches recent address translations.
- TLB organization similar to cache orgn. (direct mapped, set-, or full-associative).
- Size of TLB is small (4 – 512 entries).
- TLBs is important for fast translation.

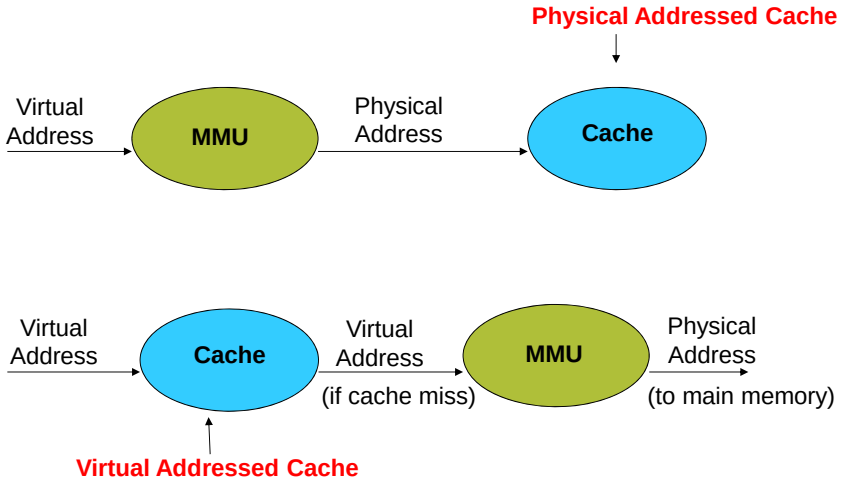
Translation using TLB

- Assume 128 entry 4-way associative TLB



13

Q1: Caches and Address Translation



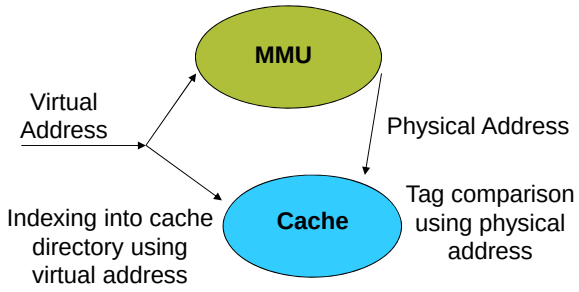
14

Which is less preferable?

- Physical addressed cache
  - Hit time higher (cache access after translation)
- Virtual addressed cache
  - Data/instruction of different processes with same virtual address in cache at the same time …
    - Flush cache on context switch, or
    - Include Process id as part of each cache directory entry
  - Synonyms
    - Virtual addresses that translate to same physical address
    - More than one copy in cache can lead to a data consistency problem

15

Another possibility: Overlapped operation



Virtual indexed physical tagged cache

16

Addresses and Caches

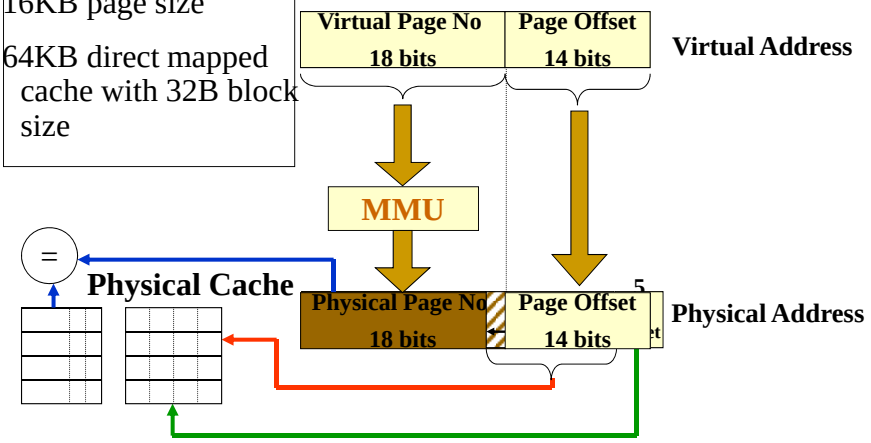
- `Physical Addressed Cache'
  Physical Indexed Physical Tagged
- `Virtual Addressed Cache'
  Virtual Indexed Virtual Tagged
- Overlapped cache indexing and translation
  Virtual Indexed Physical Tagged
- Physical Indexed Virtual Tagged (?)

17

Physical Indexed Physical Tagged Cache

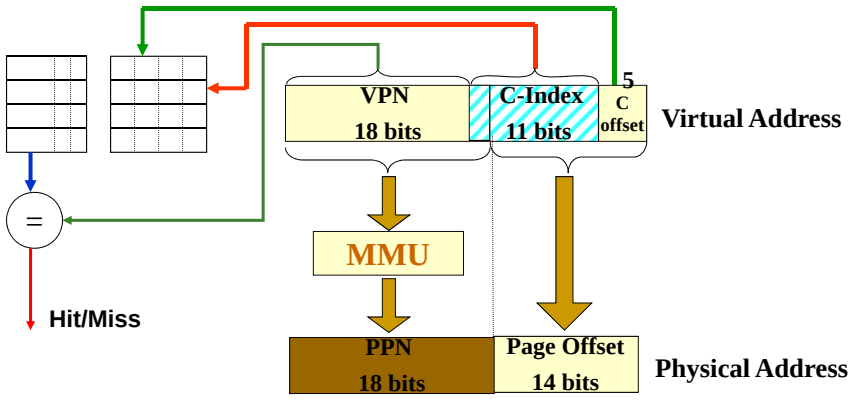
16KB page size

64KB direct mapped cache with 32B block size



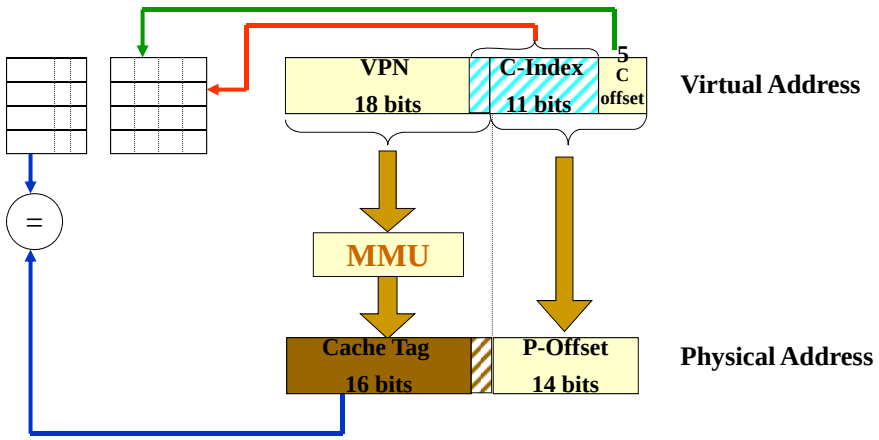
18

Virtual Index Virtual Tagged Cache



19

Virtual Index Physical Tagged Cache



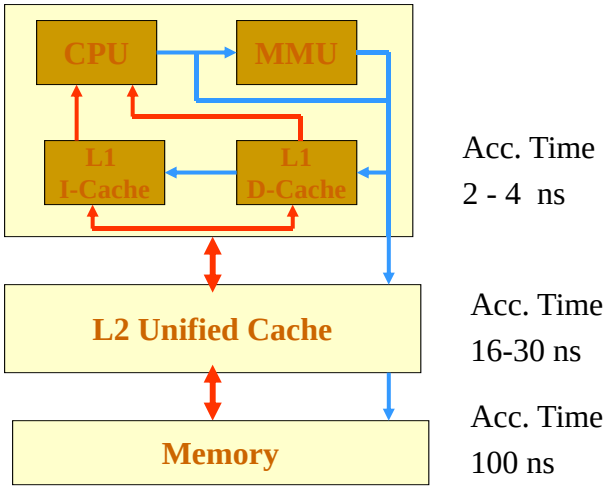
20

Multi-Level Caches

- Small L1 cache -- to give a low hit time, and hence faster CPU cycle time .
- Large L2 cache to reduce L1 cache miss penalty.
- L2 cache is typically set-associative to reduce L2-cache miss ratio!
- Typically, L1 cache is direct mapped, separate I and D cache orgn. L2 is unified and set-associative.
- L1 and L2 are on-chip; L3 is also getting in on-chip.

21

Multi-Level Caches



Acc. Time
2 - 4 ns

Acc. Time
16-30 ns

Acc. Time
100 ns

22

Cache Performance

- One Level Cache

  AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$

- Two-Level Caches

  AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$

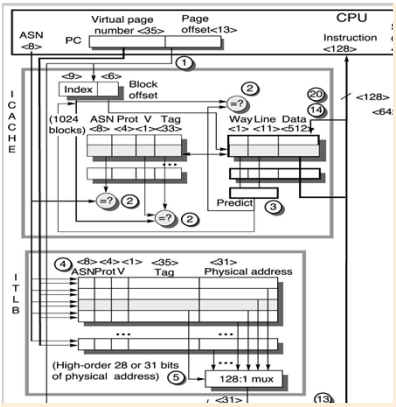
  Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$

  AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x
  (Hit Time$_{L2}$ + Miss Rate$_{L2}$ + Miss Penalty$_{L2}$)

23

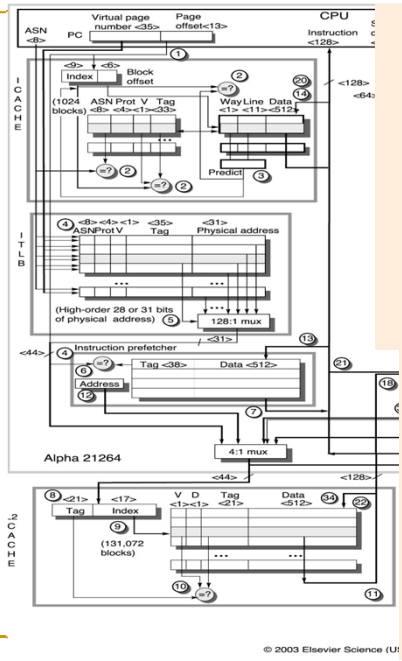
Putting it Together: Alpha 21264

- 48-bit virtual addr. and 44-bit physical address.
- 64KB 2-way assoc. L1 I-Cache with 64byte blocks (⇒ 512 sets)
- L1 I-Cache is virtually index and tagged (address translation reqd. only on a miss)
- 8-bit ASID for each process (to avoid cache flush on context switch)



Alpha 21264

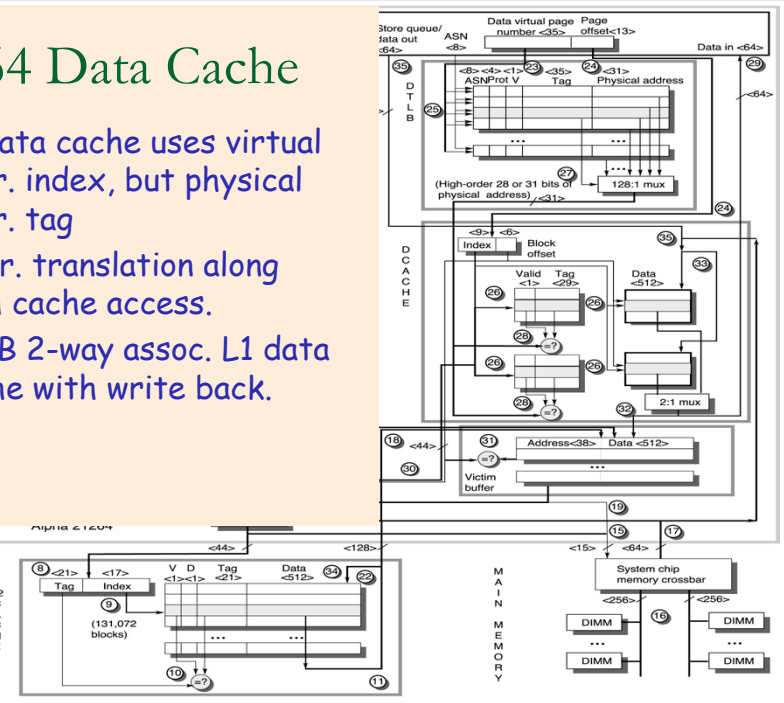
- 8KB page size (⇒ 13 bit page offset)
- 128 entry fully associative TLB
- 8MB Direct mapped unified L2 cache, 64B block size
- Critical word (16B) first
- Prefetch next 64B into instrn. prefetcher



25

21264 Data Cache

- L1 data cache uses virtual addr. index, but physical addr. tag
- Addr. translation along with cache access.
- 64KB 2-way assoc. L1 data cache with write back.



Q2: High Performance Pipelined Processors

- Pipelining
  - Overlaps execution of consecutive instructions
  - Performance of processor improves
- Current processors use more aggressive techniques for more performance
- Some exploit Instruction Level Parallelism - often, many consecutive instructions are independent of each other and can be executed in parallel (at the same time)

27

Instruction Level Parallelism Processors

- Challenge: identifying which instructions are independent
- Approach 1: build processor hardware to analyze and keep track of dependences
  - Superscalar processors: Pentium 4, RS6000,…
- Approach 2: compiler does analysis and packs suitable instructions together for parallel execution by processor
  - VLIW (very long instruction word) processors: Intel Itanium

28

ILP Processors (contd.)

Pipelined

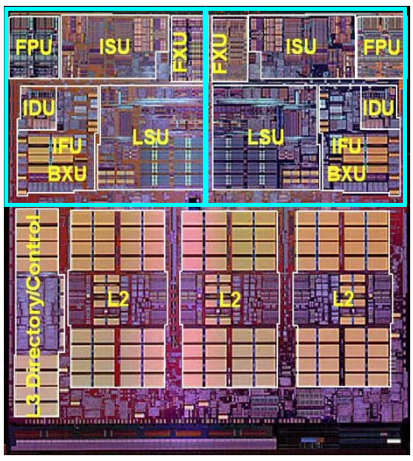
Superscalar

VLIW/EPIC

29

Multicores

- Multiple cores in a single die
- Early efforts utilized multiple cores for multiple programs
  - Throughput oriented rather than speedup-oriented!
- Can they be used by Parallel Programs?



30