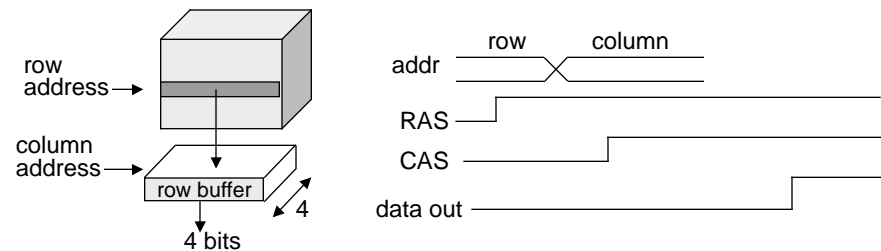

Lecture 13: Main Memory and Virtual Memory

Kunle Olukotun
Gates 302
kunle@ogun.stanford.edu

<http://www-leland.stanford.edu/class/ee282h/>

1

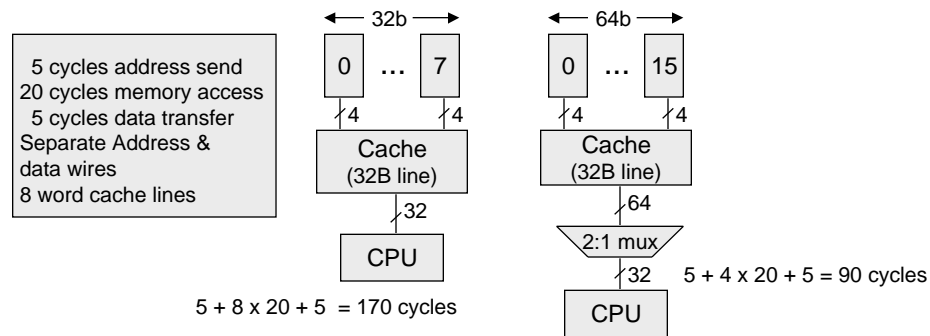
DRAM design



- Cycle time (repeat rate) is larger than access time; typical values are 120 ns and 75 ns.
- Data is stored on leaky capacitors and must be refreshed every 2ms or so by row access.
- SRAM has typically 1/16 the capacity but is 8 times faster, so is used for caches
- DRAM problem: bandwidth isn't increasing with density

2

Speeding up main memory: #1: Make it Wider

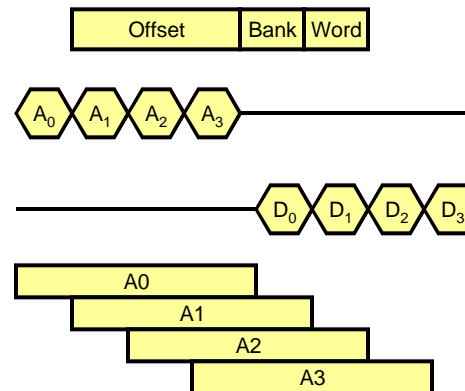


- Increase the width of the bus between memory and cache+CPU
- Advantages:
 - » Doubles bandwidth
 - » Good match for multiple-word cache lines
- Disadvantages
 - » Wider buses are more expensive
 - » Increases minimum memory increment

3

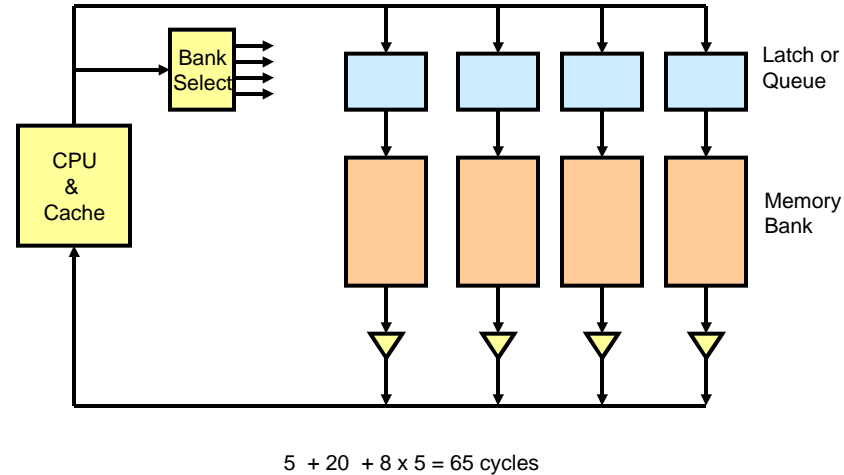
Speeding up main memory: #2: Interleaved Memory

- Distribute memory address space across memory *banks*
- Route requests to banks based on low *block* address bits
- Allows memory accesses to go in parallel



4

Interleaved Memory Organization



5

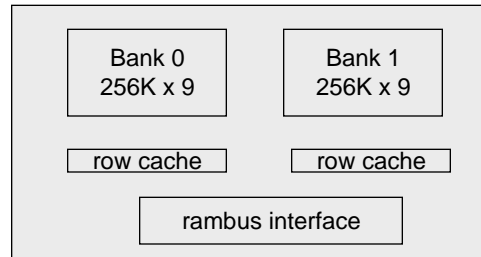
Speeding up main memory: #3a: Take advantage of DRAM design

- Conventional Techniques
 - » Page mode: Access the row once, and do repeated CAS addressing to get bits within the row
- Evolutionary techniques
 - » Synchronous DRAM (SDRAM):
 - system clock input, not just asynchronous RAS and CAS
 - pipelined internally, e.g. overlapped row access and data transfer
 - » Cached DRAM (CDRAM) (Mitsubishi):
 - SDRAM with a small cache internally
 - memory controller has the tag storage for all the chips of a line

6

Speeding up main memory: #3b: Radically new DRAM design

- RAMBUS
 - » Two independent banks internal to the chip, each with a row buffer
 - » Make the chip a “computer” that processes transactions: opcode, address, and packet size
 - » Multiple outstanding requests to different chips on the same bus
 - » Block transfers of data on both edges of a 300 Mhz clock:
1 byte every 2 ns (600 MB/s per chip).
 - » Not electrically compatible with standard parts
 - » What is the cost, will it survive



7

Fill Frequency

- Fill Frequency
- DRAM bandwidth/ DRAM capacity = MB/s*1/MB= 1/s

8

Virtual Memory: Motivation

- An address remapping scheme:
programmer virtual addresses ---> memory physical addresses
- Original motivations:
 - » Allow the programmer-visible address space to be bigger than the real memory. Avoid overlays! Keeps the hot spots in memory.
 - » Allow selective sharing of data between processes
 - » Protection
 - » Simplifies relocation
 - » Fast program startup
- Also serves as another level of cache in the hierarchy: DRAM-->disk
 - » But Huge (1–6 Mcycles) cache miss time, so emphasize low miss rate
 - fully associative: any block (“page”) goes anywhere
 - But fully associative + large size (16+ MB) means too many comparators, so use table lookup instead
 - But table is large, so use yet another cache (“translation lookaside buffer”) for the table!

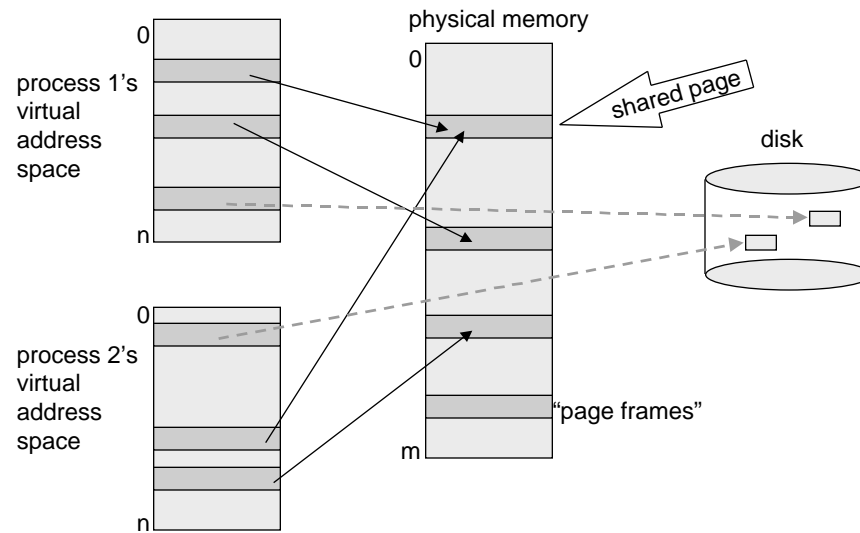
9

Virtual Memory: The basics

- Programs references “virtual” addresses in a non-existent memory space, which are translated in “real” addresses in memory
 - » Virtual address space may be bigger, the same as, or (rarely) smaller than the physical address space.
- Divide physical memory into blocks, now called “pages”; fixed size of 512 to 16MB (4K typical) bytes
 - » Holds a recently-used subset of the virtual memory
- Virtual-to-real translation: by indexed table lookup, plus a cache for recent translations
- Block placement: any block anywhere (fully associative) to reduce miss rate
- Replacment policy: LRU, or some approximation thereto
- Write policy: write-back, using a dirty bit

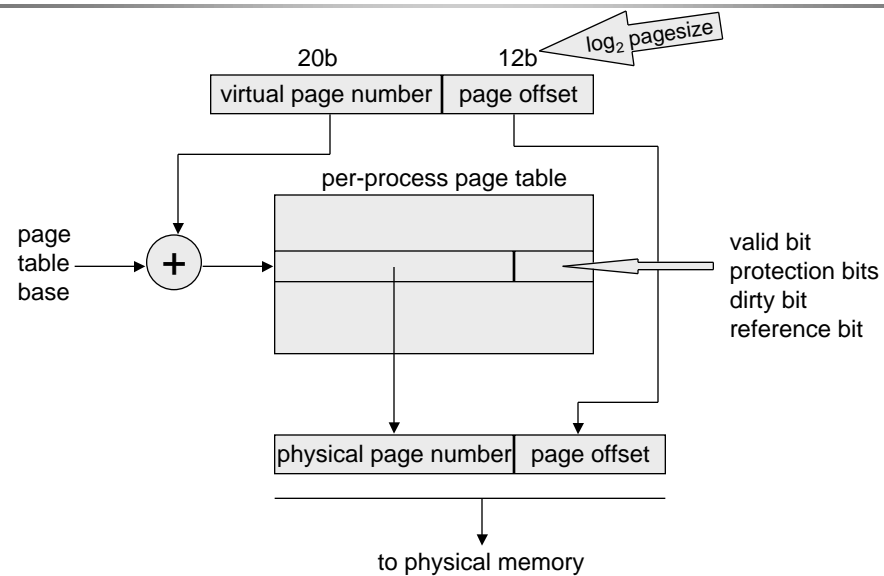
10

Virtual Memory: Page mapping



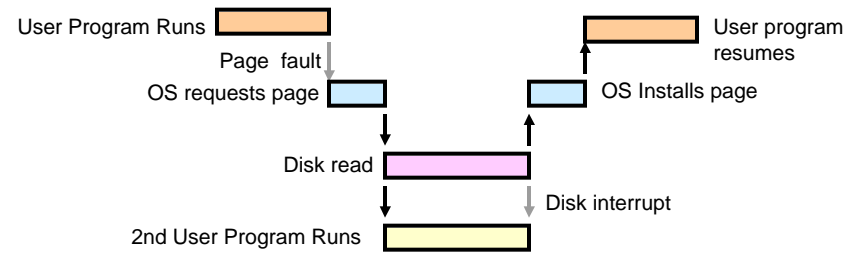
11

Virtual Memory: Address translation



12

Virtual Memory



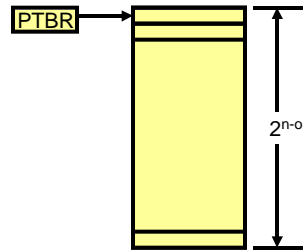
13

Selecting a page size

- Reasons for larger page size
 - » less memory for page table
 - » simplifies fast cache hit times
 - » efficient transfer to or from secondary storage
 - » Number of page table caches is limited by cycle time so larger page size maps more memory
 - » limits page table cache entries for large physical memory structures (e.g. frame buffer)
- Reasons for smaller page size
 - » less wasted storage : 1.5 X page size (code, stack, heap)
 - » quicker process start for small processes
- Hybrid solution:
 - » multiple page sizes
 - » Alpha supports 8 KB, 64 KB, 512 KB, 4 MB pages

14

Page Table Organization



- Flat page table has size proportional to size of *virtual* address space
 - » can be very large for a machine with 64-bit addresses and several processes
- Three solutions
 - » page the page table (fixed mapping)
 - » multi-level page table (lower levels paged - Tree)
 - » inverted page table (hash table)

15

Multi-level Page Table

- “64 b” address divided into 3 segments
 - » seg0 (b63=0) user code/heap
 - » seg1 (b63=1, b62=1) user stack
 - » kseg (b63=1, b62=0) kernel segment
- 3 level page table
 - » one page each
 - » only 43 unique bits of VA
- PTE bits
 - » valid
 - » kernel read & write enable
 - » user read & write enable

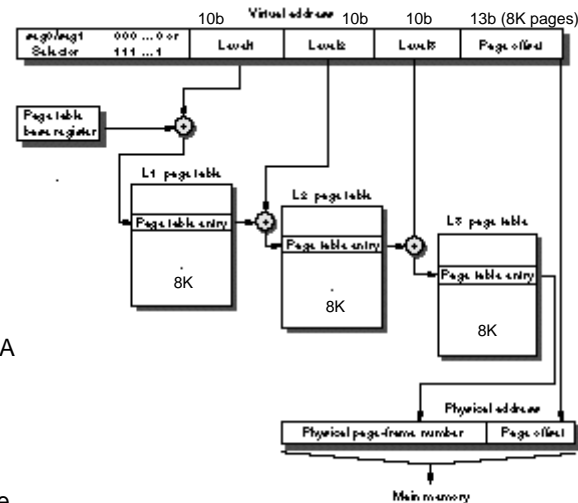
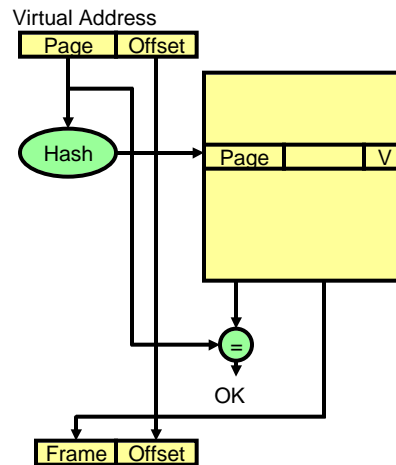


Fig 5.43, p. 451

16

Inverted Page Tables



- Store only PTEs for pages *in* physical memory
- Miss in page table implies page is on disk
- Need KP entries for P page frames (usually $K > 2$)
- Page table organization can be defined by software if TLB miss raises an exception

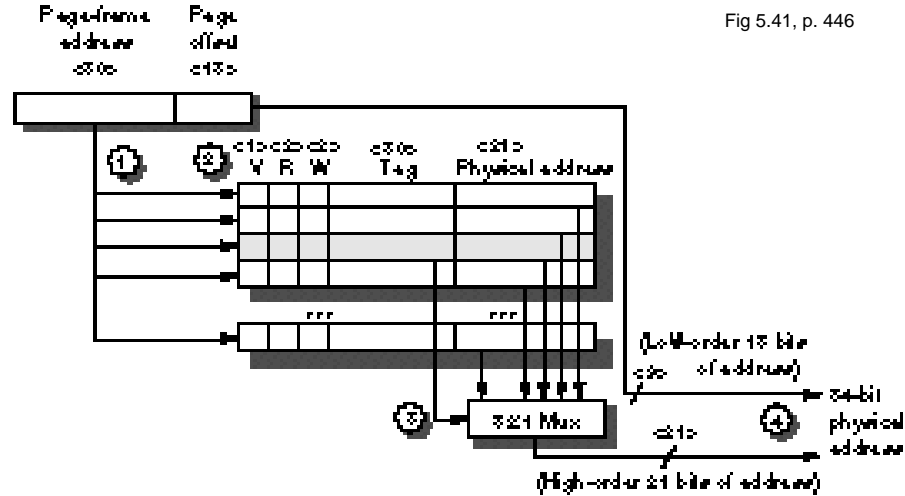
17

Address translation realities

- The translation process using page tables *takes too long*.
- Use a cache of recent translations: “translation lookaside buffer” (TLB) or “address translation cache” (ATC)
 - » Typically 8-1024 entries
 - » Only holds translations for pages in main memory
 - » 1 cycle hit time
 - » Highly or fully associative
 - » Miss rate <1%
 - » 10 to 50 cycle miss penalty
 - Can be processed by hardware lookup of page tables (IBM, x86)
 - Can be handled in software (MIPS)
 - Page table structure and algorithms is not fixed by the processor
 - » Must get purged on a process context switch, or the TLB entries must be marked with an ASID (more later)
 - » Need two TLBs -- one for instructions, one for data?

18

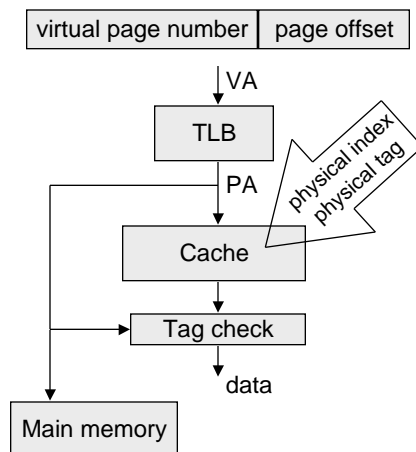
Virtual Memory: Translation Lookaside Buffer (TLB)



19

Virtual memory and caches: Real-address cache

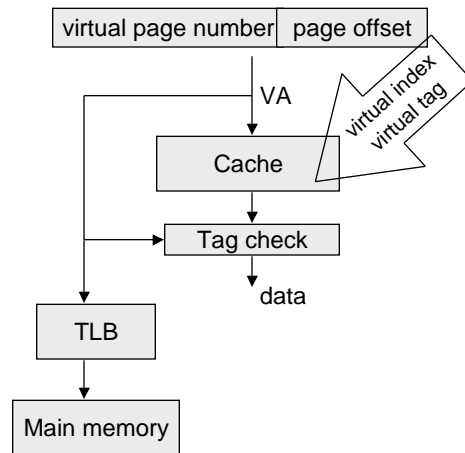
- It would be nice if the TLB did virtual-to-real translation before the cache is accessed:



- Transparent to software: it just makes memory faster
- Easy to control sharing of virtual addresses
- But it's a serialized operation: requires two cycles
- Used by VAX 11/780

20

Virtual memory and caches: Virtual-address cache



- An alternative is to cache using virtual addresses only.
- One cycle for cache hit!
- Need to do address translation only on miss
- But....

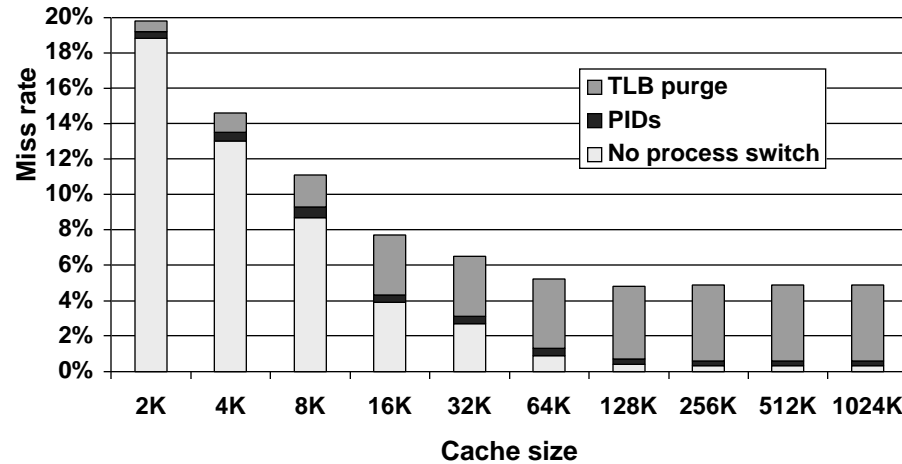
21

Virtual memory and caches: Virtual-address cache problems

- Problems:
 - » Can't distinguish virtual addresses from different processes
 - Flush the TLB when process switch occurs?
 - Include an ASID (address space ID) as part of the cache tag, and flush only when ASIDs are reassigned.
 - » Aliases (the same data addressed using different virtual addresses) will appear twice in the cache.
 - Hardware anti-aliasing: force all cache blocks to have a unique physical address (hard)
 - Software anti-aliasing: restrict the addresses of shared blocks so that only one copy can be in the cache
 - » I/O devices use physical addresses and would have to remap to virtual addresses to use the cache
 - » Shared-memory multiprocessors require invalidating our cache entries based on physical addresses ("snooping"). Would also need to remap to virtual addresses, or have both physical and virtual tags in the cache
- Used by: SPARC (has address restrictions)

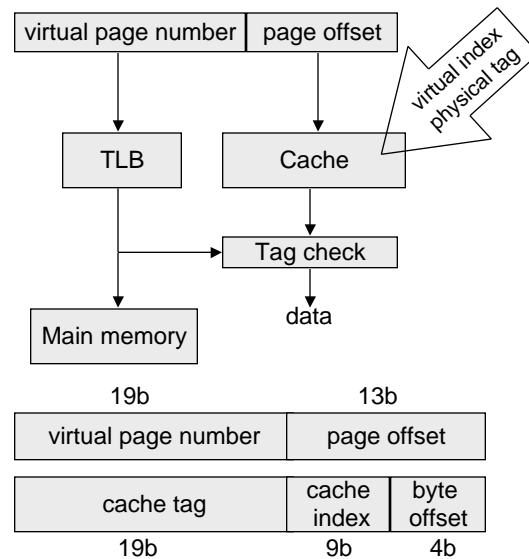
22

Virtual memory and caches: Miss rate of virtual caches



23

Virtual memory and caches: TLB and Cache in parallel

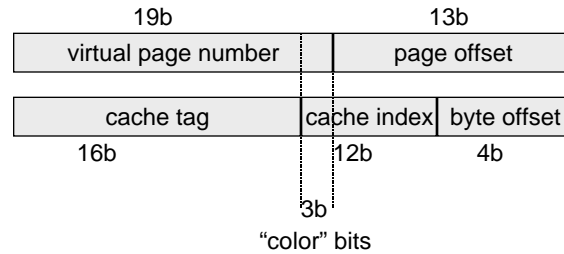


- Translation and cache access in one cycle
- Cache data is based on physical addresses; no problems with:
 - » aliasing
 - » I/O
 - » snooping
- But: only works when the VPN bits are not needed for TLB lookup
- cache size \leq page size * associativity
- Most common solution

24

Virtual memory and caches: TLB and Cache in parallel, part 2

- If a few of the VPN bits are needed for the cache lookup, all is not lost.

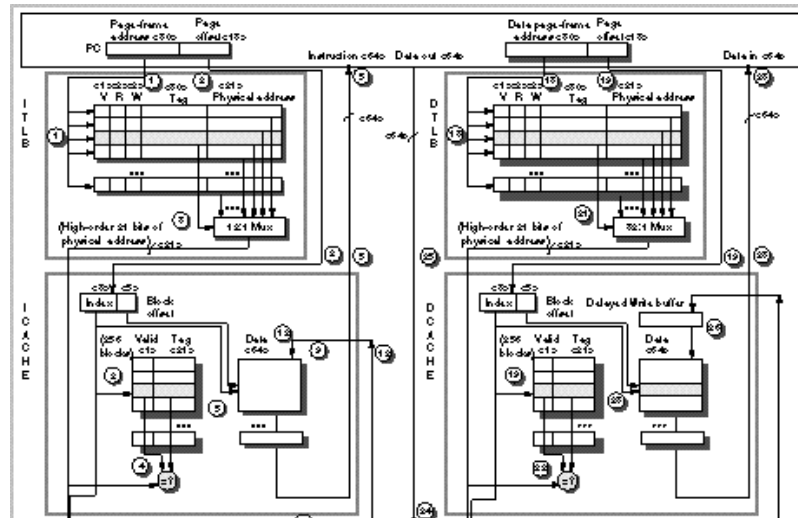


Each page frame in main memory is of a "color" depending on the value of the color bits. The OS can allocate page frames (physical addresses) that match the virtual address color, by having multiple linked lists.

25

Alpha 21064 Memory Hierarchy

Fig 5.47, p. 462



26

Alpha 21064 L2 Cache Miss

- L2 Cache miss
 - » Step 14: Send read request to main memory
 - » Step 15: Put dirty victim block in victim buffer
 - » Step 16: Load new block in L2 cache 16B at a time
 - » Step 17: Write victim buffer to memory
- Data loads are like instruction fetches, except use the DTLB and D-cache instead of the ITLB and I-Cache
- Allows hits under miss
- On a read miss, the write buffer is flushed first to avoid RAW hazards

29

Alpha 21064 Data Store

- Data store
 - » Step 18: DTLB lookup and protection violation check
 - » Step 19: D-Cache lookup (8KB, DM, writethrough)
 - » Step 22: Check D-Cache tag
 - » Step 24: Send data to write buffer
 - » Step 25: Send data to delayed write buffer in front of D-cache
 - (write hits are pipelined)
 - » Step 26: Write previous delayed write buffer to D-cache
 - » Step 27: Merge data into write buffer
 - (coalesced write buffer)
 - » Step 28: Write data at the head of write buffer to L2 cache (15 cycles)

30

Alpha 21064 Cache Performance

Benchmark	Stall penalty in CPI						Miss rate		
	L1-I	L1-D	L2	Cache total	Other	Total	L1-I	L1-D	L2
SPECint92	.13	.20	.02	.35	1.51	1.86	1.8%	13%	0.6%
SPECfp92	.06	.38	.01	.45	1.69	2.14	0.85%	20.9%	0.27%
Commercial	.41	.42	.66	1.49	2.31	3.8	5.9%	32.3%	10.3%

Commercial = transaction processing, sorting

31

Conclusions

- CPU performance outpacing main memory performance
- Principle of locality saves us : memory hierarchy
- The memory hierarchy should be designed as system
- Key old ideas
 - » bigger caches
 - » higher set-associativity
- Key newer ideas
 - » non-blocking caches
 - » dynamic scheduling
 - » multi-port caches
 - » software controlled prefetching

32