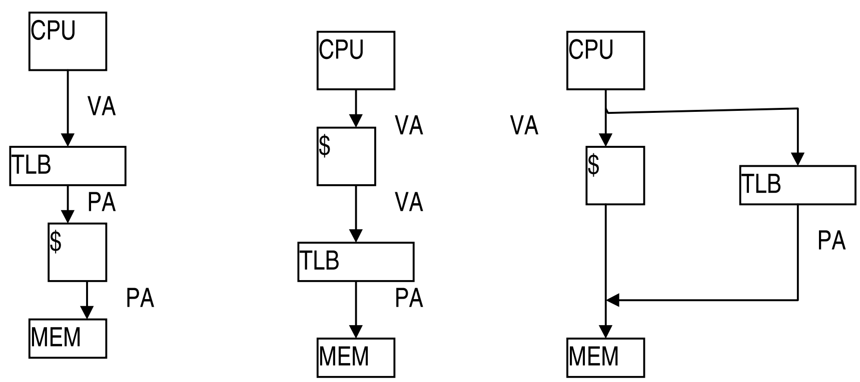
Lecture #18

****
Review -- 1 min
****
Reducing miss penalty
Reducing Hit Time
    KISS
    address translation, checking hit is main challenge



Background: goal is to translate from virtual address to phys address space

Q: What was the problem with #1?
#2?
We didn't talk much about #3…

1. virtually addressed, physically tagged cache
   (pipleline:
   1. start read cache[VA]; start PA = TLB(VA)
   2. get data and physical tag; check physical tag against PA

(This is picture 3 above)
QUESTION: What is DA?
ANSWER: cache index < page size

→ sometimes use set associative to get around this
e.g. IBM 3033 – 16-way set assoc to allow 64KB
physically tagged cache with 4K pages.

****
Outline - 1 min
****
DRAM
Memory Organizations to Improve Main Memory
Case Study: 21064

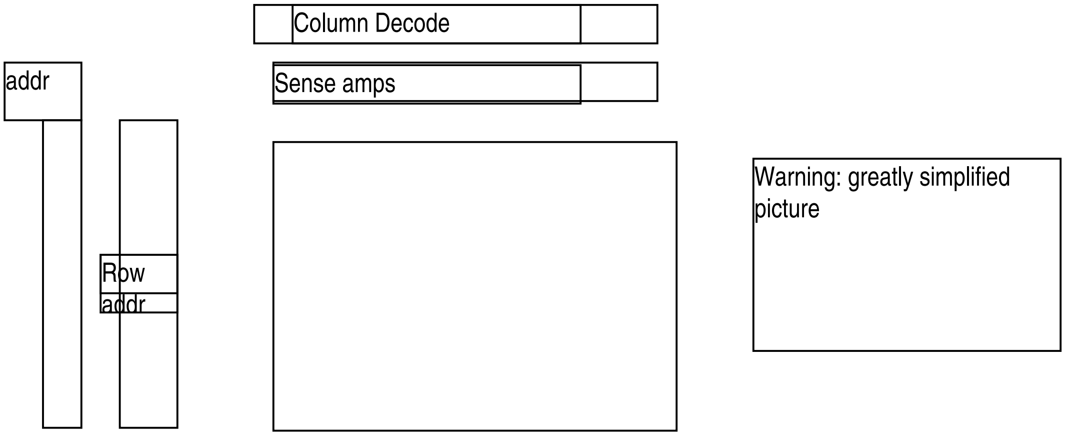
****
Preview - 1 min
****
M: Discussion
W: Q&A, Exam
F: No class

****
Lecture - 20 min
****
DRAM v. SRAM
Caches are SRAM (6 transistors per bit)
Main memory: DRAM (1 transistor per bit)
(Size advantage > 6x b/c optimize process for size not speed)

DRAMs optimized for cost, capacity,
DRAM Characteristics

1. need to refresh (~8ms)
2. split address (to save pins) RAS, CAS
3. high latency
   (split address, sense amps to get meaning from 1 transistor)

Picture:



4. High internal bandwidth?
   (square root of bits per RAS) → for 4 mbit generation
   2 mbit per ~100ns → hundreds of gigabits per second

Efforts to improve main memory have focused on BW
QUESTION: that may still help latency – why?
(allows more aggressive prefetching; also: less queuing)

Example 1: Wide main memory
(draw picture)

Example 2: Interleaved Banks
(conceptually same as wide data paths –
access wide memory in parallel, but
serialize sending data across narrow bus)

Example 3: Independent Banks
(draw picture)
conceptually – a bunch of independent memories

QUESTION: compare Interleaved banks with independent banks – which will be better for what?

(which is better for sequential access? Which is better for simultaneous random access? Which is cheaper?)

One challenge with banks: *Avoiding Bank Conflicts*

stride = #banks * K
→ every access goes to same bank
stride = #banks / K
→ every K'th access goes to same bank

solution:

1. sw: prime-sized strides
2. hw: prime # of banks
   (But wait – division is hard – it takes several cycles
   → chinese remainder theorum)

Question: how many banks?
Sequential access proceeds w/o stall as long as #banks >= #clocks to access a bank

Problem: DRAMs per system falling over time
increasing size → fewer chips → harder to have banks

(capacity per DRAM 4x/3 years
capacity per system: somewhat more slowly)

	DRAM Generation					
	'86	'89	'92	'96	'99	'02
	1Mb	4Mb	16Mb	64Mb	256Mb	1Gb
4MB	32	8				
8MB		16	4			
16MB			8	2		
32MB				4	1	
64MB				8	2	

128MB	4	1
256MB	8	2

****
Admin - 3 min
****
HW3 due today
Exam Wednesday

****
Lecture - 24 min
****

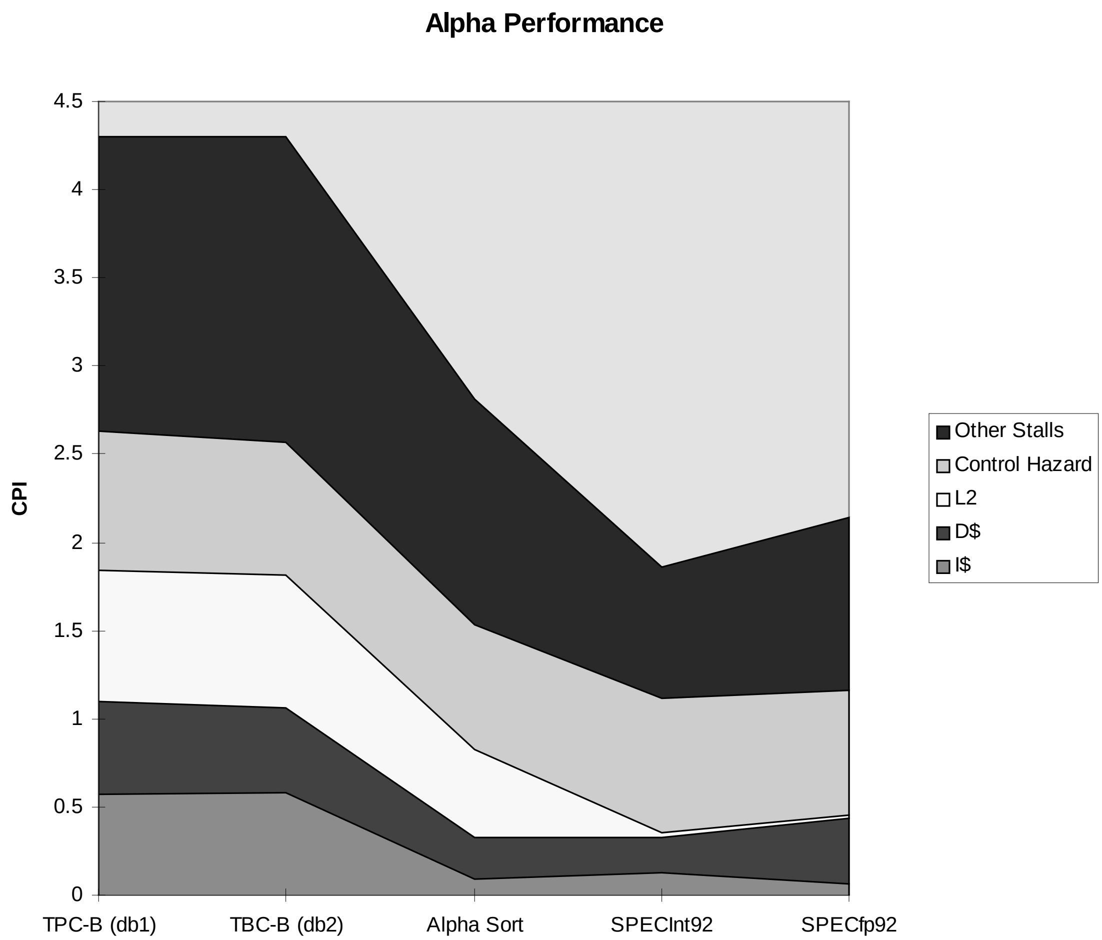
Case Study: 21064 memory

<transparency: figure 5.47 >
Stats: 8 entry ITLB; 32 entry DTLB (variable sized pages)
8K L1 I$; 8K L1 D$ (32-byte block)
2MB L2 $ (32-byte block)

1. What happens on instruction read
hit? Miss (stream buffer)? TLB Miss (PAL code)? Main memory miss?
QUESTION: on main memory miss – what happens to physically tagged items in cache (OS has to flush?)
2. What happens on data read
Hit (virtual index, physical tag)?Miss?TLB Miss?
3. what happens on data write
TLB access check; L1 is write through (wait for TLB check!) w. write buffers; L2 is write back; what happens on L2 wb (victem buffer)?

Performance of 21064

<transparency: figure 5.49>



Cross-cutting issues

- Superscalar CPU and # cache ports
- speculative execution and non-faulting option on memory
- parallel execution v. cache locality

want far separation fr independence v. want reuse of data to avoid cache misses

- I/O and consistency between cache and memory

    caches→ multiple copies of data
    consistency by HW or SW?
    where to connect I/O to computer?

PITFALLS
1. Predicting Cache perf from different programs
(ISA, compiler)
<fig 5.50>
2. simulating too small an address trace
<fig 5.51>

****
Summay - 1 min
****
Memory Organizations to Improve Main Memory
Virtual Memory
Case Study: 21064

]

OUTTAKES

Virtual Memory

Purpose:

1. mapping from VA (2^32 – 2^64) to PA (2^30—2^40)
2. protection
   interprocess
   read/write/execute
3. page to disk

1. 4Q's
2. TLB
3. TLB Misses

1. 4 Q's of cache – dominated by large size, slow disk

1. Block placement
   fully associative (b/c miss penalty high)
2. locating a block

(next)
3. block replacement
LRU (approximated)
4. write policy
   write back

2. TLB

<show how tlb is related to each of above>

1. Page placement
2. Locating a page

3. Block replacement –pseudo LRU
   "use" bits
4. write policy – write back
   dirty bits

Other issue: protection
   permission bits

Other issue: variable sized pages (e.g. 8K-4MB on alpha)
   → page size field; only compare subset of tag bits for hit

3. TLB is a cache – what happens on misses?
(describe page table)

Read address, flags from memory
What address of memory? (Can't use the address – that's the address of the page not the page mapping)

→ page table

    mapping address→page entry for address

    Lookup: what are our options? Not too many data structures in the world:

- linked list

  64MB/8K pages = 8000 pages – search too slow

- array

  QUESTION: what's the problem

  lookup is by VA – 2^64 bits of VA / 8K = page table is 2^51 bits long

  → too big

- hash table

  IBM Unix used this for a while (maybe still do)

  "inverted page table"

- tree

  most common – Solaris, Digital Unix

Example – DEC Alpha
3-level page table
Each node is 1 8KB page of 8-byte addresses – <= 1024 entries per node
43-bit virtual address – 3x10 bit page selectors, 13-bit offset

TLB Miss penalty: 20 cycles (avg)

Selecting a page size

Mapping is per-page
Question: Arguments for a larger/smaller page

larger

- page table inversely proportional to page size → saves memory
- fast hit tames when cache <= page size (VA cache)

bigger pages makes bigger cache more feasible

- xferring larger page from disk or network mor efficient
- # tlb entries restricted by clock cycle time → larger page maps more memory→ larger page reduces TLB misses

smaller

- fragmentation – don't waste storage
- quicker process start for small processes (?)

hybrid solution: multple page sizes
-----
e.g Alpha: 8K, 16K, 32K, 64K