Lecture #18

****
Review -- 1 min
****
Caches

- 4 approaches to reduce misses
- 5 Ways to reduce miss time

****
Outline - 1 min
****
Caches

- 4 Ways to improve hit time

Main memory

- Background
- System Organization
- DRAM Organization

****
Preview - 1 min
****
F: VM and Case Study
M: IRAM

****
Lecture - 20 min
****

Comments on Burger's talk.

Take home message:

1. cache hit time sets cycle time
2. Not so much reading the data as deciding
   you've got the *right* data

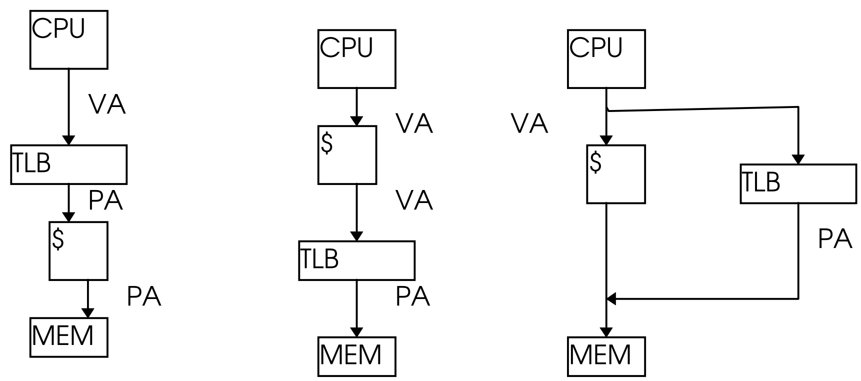
ways to improve hit time
1. small, simple caches

last time

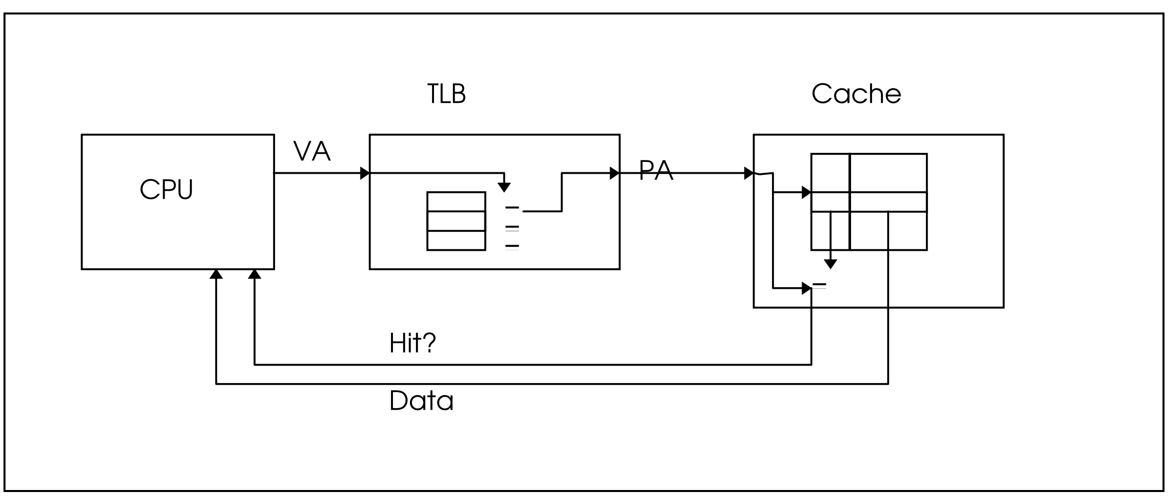
2. pipeline read hits
   last time

3. pipeline write hits
   last time
4. avoid address translation

CPU uses *Virtual Addresses*
Mem System uses Physical Addresses
TLB Translates from VA to PA



Traditional: 2 serial cache lookups on critical path!



Approach 2: Virtually Addressed Cache

2 problems

1. aliases
   solutions
   1. page coloring
   → page coloring – if 2 VA's map to same PA then lower N bits will match
   → both addresses map to same cache line

   (Still get cache miss for one of the addresses, but better than getting wrong answer on a write)

   2. 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.

2. cache flush on process switch
   problem: different processes use same VA for different things
   naïve solution: flush cache on proc. Switch (hurts performance)
   → add process identifier tag – only a hit if a match

QUESTION: how would you implement process identifier tag
(main ideas – set a PID register in CPU on context switch
design goal—minimize bits of storage → use few bits
to cover common case of a couple processes running at same time (flush cache if recycle PID))

****

Admin - 3 min

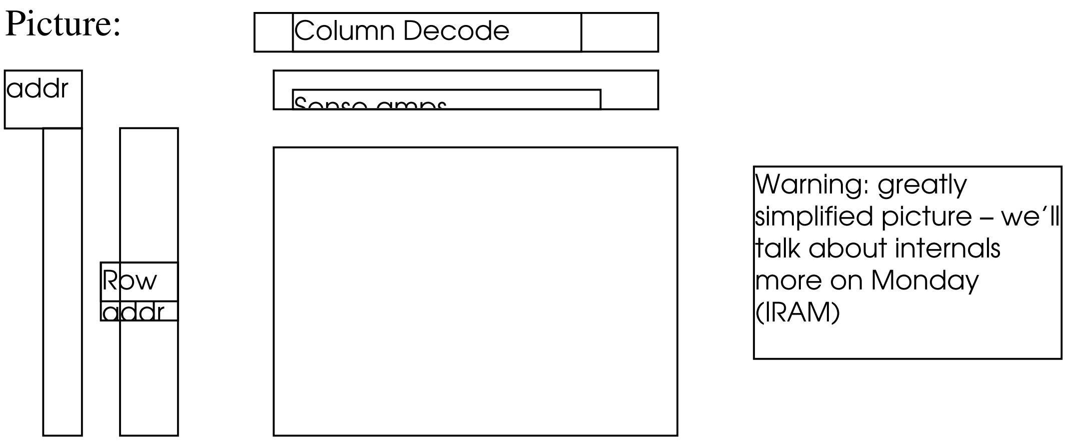
****
Project proposals

Wednesday – Q&A
Wednesday Night – exam 6-9PM Tay 3.144

****
Lecture - 24 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?)

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

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)

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

Second Approach: Tap Internal DRAM BW

"page mode", "static column mode" – read sequential bytes from internal buffer

EDO – extended data out – same idea; adds twist – can start next address while still reading current buffer

RAMBUS – same idea + fancy bus → 2ns per byte = 500 MB/s per chip!
(but 30% cost premium for 10x BW in cost-sensitive market)

****
Summay - 1 min
****

Fast Hits:

KISS
pipeline
get address translation oout of the way.

Fast main memory:
- BW – reasonably easy
- latency – still a problem

?Will we hit the memory wall?