EE282
Computer Architecture

Lecture 11-12:
Caches

November 6-8, 2001

Andrew Wolfe
Computer Systems Laboratory
Stanford University

The Memory Bottleneck

- Typical CPU clock rate
  - 500MHz (2ns cycle time)
- Typical DRAM access time
  - 60ns (about 30 cycles)
- Typical main memory access
  - 200ns (100 cycles)
    - DRAM (60), precharge (40), chip crossings (50), overhead (50).
- Our pipeline designs assume 1 cycle access (2ns)
- Average instruction references
  - 1 instruction word
  - 0.3 data words
- This problem gets worse
  - CPUs get *faster*
  - Memories get *bigger*
- Memory delay is mostly communication time
  - reading/writing a bit is *fast*
  - it takes time to
    - select the right bit
    - route the data to/from the bit

Economics of Memory

Cost of fast memory (n$/bit) ____ Cost of slow memory

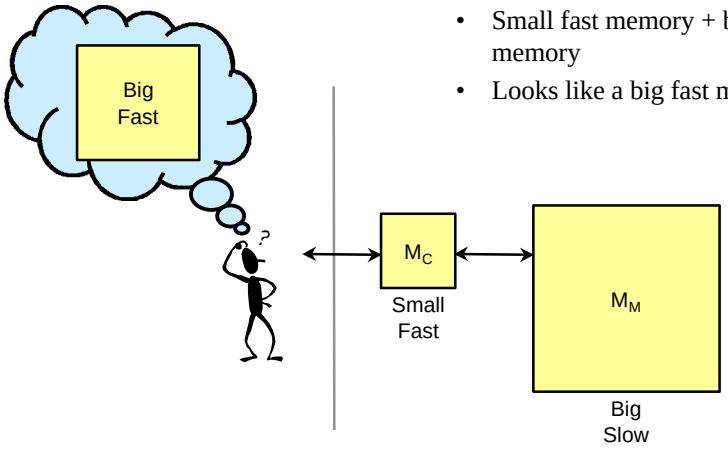
Cost of large memory ____ Cost of small memory

Physics of Memory

Access time of large memory ____ Access time of small memory

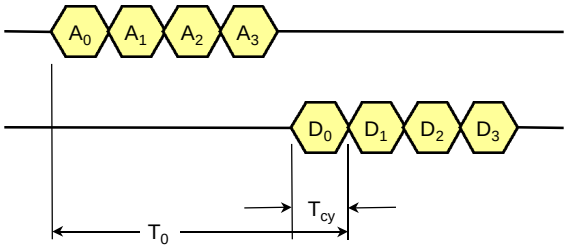
Cost of fast memory (fJ/bit-access) ____ Cost of slow memory

Cache Memory



- Small fast memory + big slow memory
- Looks like a big fast memory

Memory System Performance



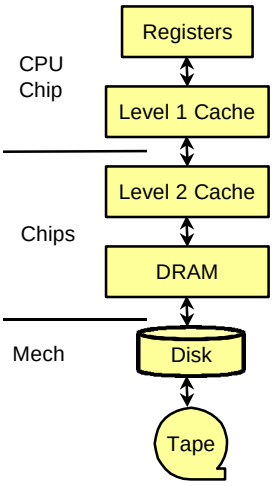
Latency, T_0

$T_0 + k*T_{cy}$ for multiple words

Throughput, $1/T_{cy}$

Bandwidth, W/T_{cy}

The Memory Hierarchy



Registers	1cy	3-16 words/cycle	compiler managed
Level 1 Cache	1-3cy	1-4 words/cycle	hardware managed
Level 2 Cache	3-10cy	1-2 word/cycle	hardware managed
DRAM	30-200cy	0.3-2 words/cycle	OS managed
Disk	10^6-10^7cy	0.01 words/cycle	OS managed

Memory Technology

SRAM (Static RAM)

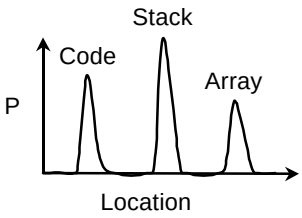
- Medium Density
  - ~10^5 b/mm^2
- Simple Timing
- Low latency
- High bandwidth
- High operating power
- Low standby power
- No refresh
- Easy to multi-port

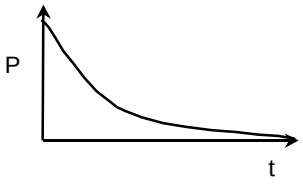
DRAM (Dynamic RAM)

- High density
  - ~10^6 b/mm^2
- Complex timing
- High latency
- High bandwidth
- Lower operating power
- Higher standby power
- Refresh required
- More challenging to multi-port

Locality of Reference

- Spatial Locality
  - likely to reference data *near* recent references
- Temporal Locality
  - likely to reference the same data that was referenced recently





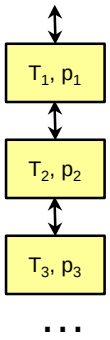
Program Behavior

- Locality depends on type of program
- Some programs 'behave' well
  - small loop operating on data on stack (towers of hanoi)
- Some programs don't
  - frequent calls to nearly random subroutines
  - traversal of large, sparse data set
    - essentially random data references with no reuse
    - reuse patterns too long or complex to manage
  - streaming data
  - this is more typical of *real* applications

Average Memory Access Time

$$T_{AVG} = pT_H + (1-p)T_M$$

$$T_{AVG} = \sum_i p_i T_i$$



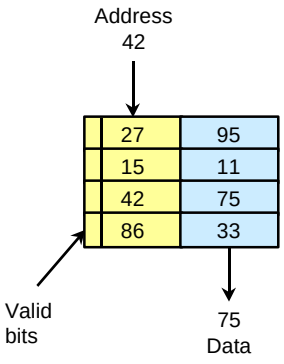
Example

T_1 = 1 cycle, p_1= 0.95

T_2 = 50 cycles

T_{AVG}= 0.95(1) + 0.05(50) = 3.45cycles

Cache Organization: Associative Memory

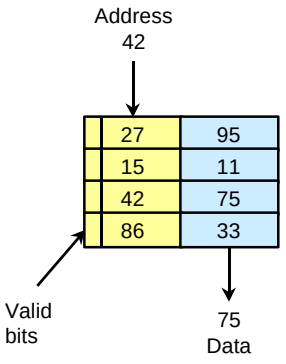


- Search *directory* memory for matching address
- Read corresponding data location
- Can put any address in any location (flexible)
  - minimizes conflict misses
- Requires a *comparator* for each address (expensive)
  - 9T or 10T CAM cell vs. 6T RAM cell

Review

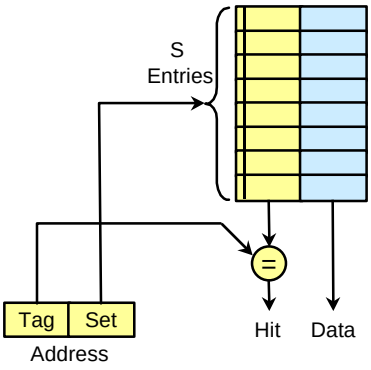
- Caches let small fast memories look like large fast memories
- Key questions:
  - How to find items in the cache
  - What to put in the cache
- Performance Factors:
  - Cache size
  - Associativity
  - Block size/Subblock size
  - Write-hit policy/Write-miss policy
  - Miss penalty/Fill rate
  - Replacement policy

Cache Organization: Associative Memory



- Search *directory* memory for matching address
- Read corresponding data location
- Can put any address in any location (flexible)
  - minimizes conflict misses
- Requires a *comparator* for each address (expensive)
  - 9T or 10T CAM cell vs. 6T RAM cell

Direct-Mapped Cache

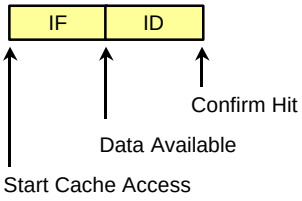


No choice about where to put data ==> conflict misses

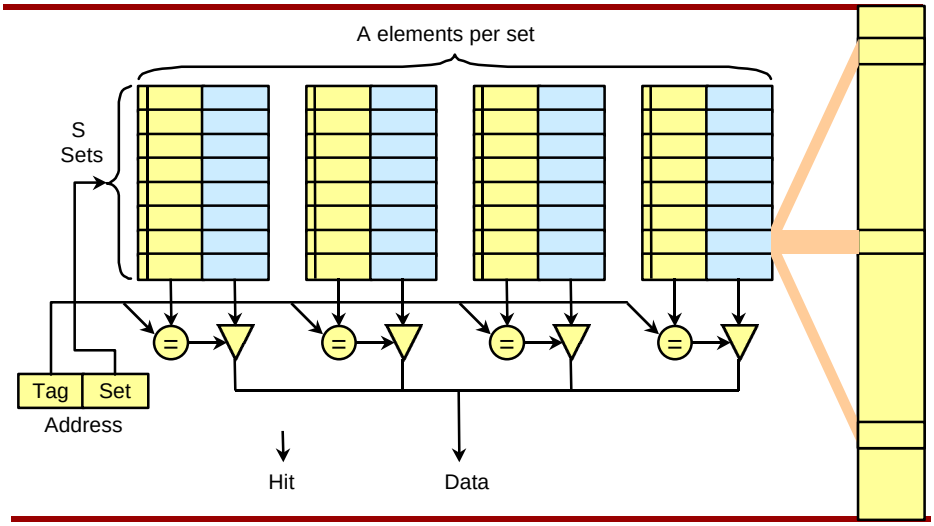
Ties number of sets to cache size - may make parallel translation hard

No switching of output data,

Can speculatively use data while performing comparison



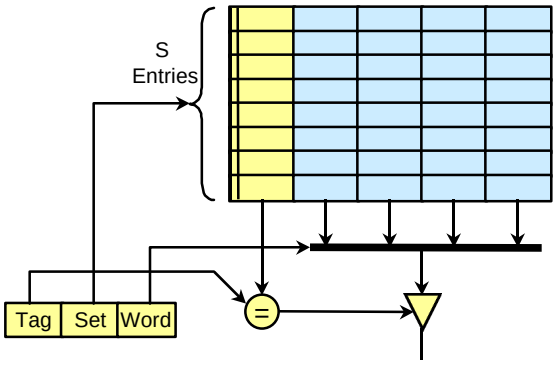
Set-Associative Organization



Set Associative Cache

- S - sets
- A - elements in each set
  - A-way associative
- In the example, S=8, A=4
  - 4-way associative 32-entry cache
- All of main memory is divided into S sets
  - All addresses in set N map to same set of the cache
    - Addr = N mod S
    - A locations available
- Shares costly comparators across sets
- Low address bits select set
  - 3 in example
- High address bits are *tag*, used to associatively search the selected set
- Extreme cases
  - A=1: Direct mapped cache
  - S=1: Fully associative
- A need not be a power of 2

Block Size



Really two types of blocks

B_A Unit of associativity - one tag for B_A words

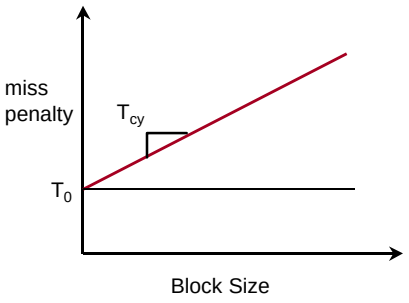
B_T Unit of transfer, move B_T words to/from main memory as a unit.

one valid bit for each B_T words

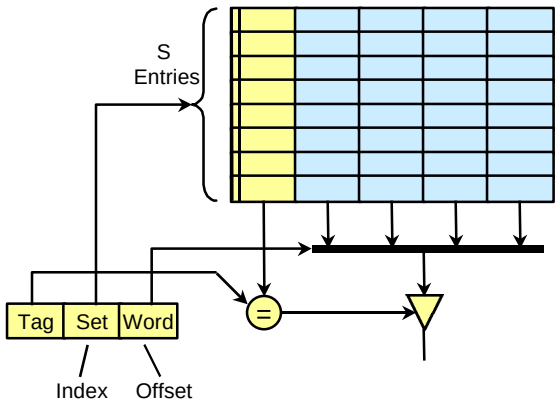
Need not be the same, but $B_A \geq B_T$

Block Size

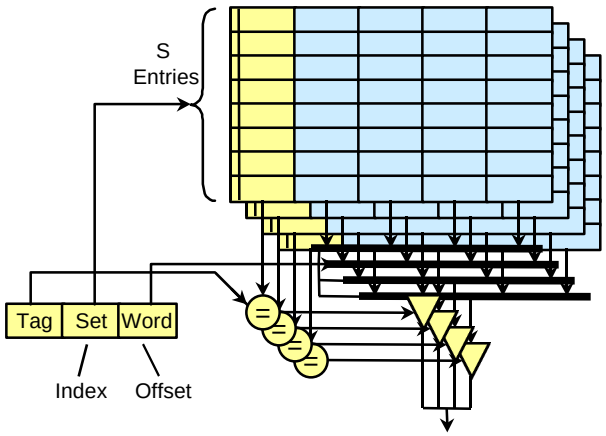
- Choice of block size based on
  - spatial locality
    - bigger blocks give lower miss rate (diminishing returns)
  - ratio of miss latency to cycle time
    - T_0/T_{cy}
    - big blocks increase miss penalty
  - cost of directory
    - larger blocks imply smaller directory
    - can be important if directory is on-chip and data array is off-chip



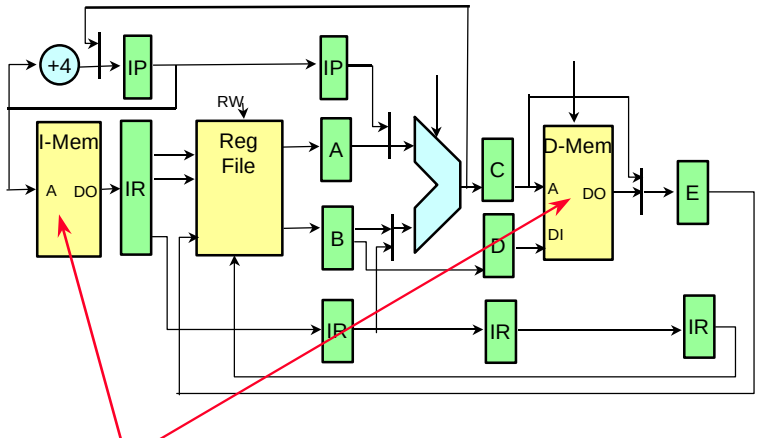
Locating Data in the Cache



Locating Data in the Cache



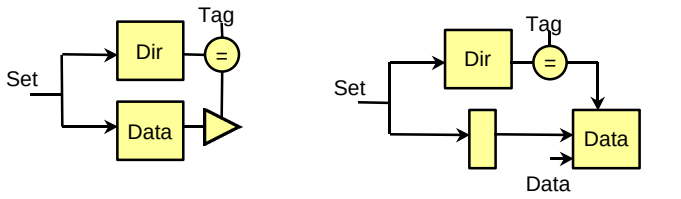
Cache Bandwidth



Cache Misses - the 3 Cs

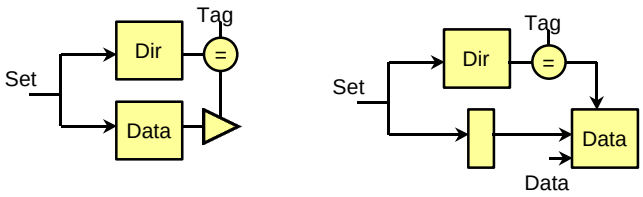
- Compulsory
  - first access to a block
    - cold start
    - non-stationary behavior
- Capacity
  - misses caused by limited cache size
- Conflict
  - misses caused by limited entries per set
- What cache parameters affect which of the Cs?

Reservation Problem



Address Bus A0 A1 A2 A3 A4 A5
Data Bus D0 D1 D2 D3 D4

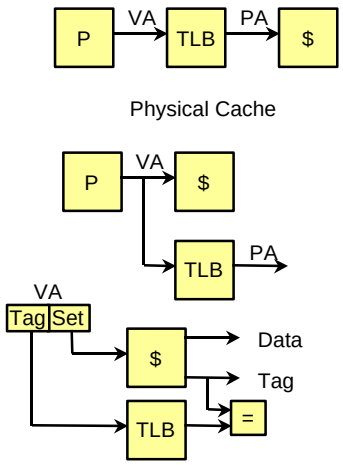
Cache Write Buffers



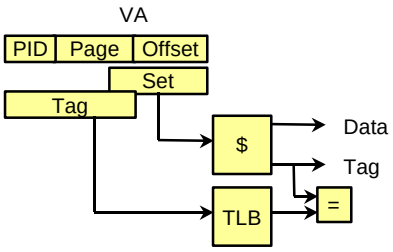
Address Bus	**A0**	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	
Data Bus	**D0**	**D1**	**D2**		**D3**	**D5**	**D6**	**D4**	**D7**
Write Buffer						***D4***	***D4***		

Mapping and Tagging

- Can address the cache (select a set) with a virtual or physical address
- Can compare virtual or physical tags
- Physically mapped, virtually tagged
  - with paging low bits are untranslated
  - what if the number of sets is greater than the blocks per page?
  - what about aliases (synonyms)? homonyms?



Mapping and Tagging II



Can overlap bits of set and tag to allow number of sets greater than page size (in blocks)

Include PID in tag to eliminate homonyms

Synonyms (aliases) still a problem unless a physical cache is used

Synonyms & Homonyms

- Memory is always physically addressed - problems can arise when caches are addressed differently
- Homonyms
  - The same VA is mapped to two PA (2 processes)
    - Problems if the cache is virtually-tagged
    - Solved by adding PID to VA
- Synonyms (aliasing)
  - Two VA are mapped to the same PA
    - Both addresses must access the same cache data for correctness.
    - Solved by:
      - Disallowing shared pages (limits system capabilities)
      - Flushing cache between processes
      - Limiting cache bank size to no more than 1 page (for VI/PT)
      - Restricting VA->PA mappings so index bits are identical (for VI/PT)

Write Hit Policy

- write *through*
  - update next level on every write
  - cache is always *clean*
  - lots of traffic to next level (mostly writes)
- write *back*
  - write to cache and mark block dirty
  - update main memory on eviction
  - less traffic to next level, but more complex eviction and coherence
- reservation problem
  - reads use directory and data array at the same time
  - writes use directory first, then data array
  - how do we pipeline to allow one read *or* write per cycle?

Write Miss Policy

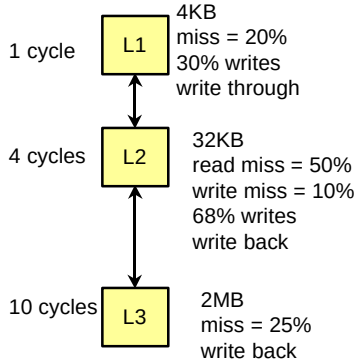
- Write allocate
  - allocate a new block on each write
  - *fetch on write*
    - fetch entire block
    - then write word into block
  - *no-fetch*
    - allocate block but don't fetch
    - requires valid bits per word
    - more complex eviction
- Write no-allocate
  - don't allocate a block if its not already in the cache
  - write *around* the cache
  - typically used by write through since we need update main memory anyway
- Write invalidate
  - instead of update for write-through
- Sometimes we would like to have a *read no-allocate* as well
  - irregular accesses on a machine with a large block size

Replacement Policy

- On a cache miss we need to *evict* a line to make room for the new line
- In an A-way set associative cache, we have A choices of which block to evict
- Which block gets booted out?
  - random
  - least-recently used
  - pseudo LRU

Multi-Level Caches

- Multiple caches often used in a hierarchy to give the effect of a large fast cache.
- Higher levels see just the misses (or write throughs) from lower levels
  - problem addresses
  - low access frequency
  - high miss rates
- Rule of thumb
  - increase cache size by 8 to halve miss rate



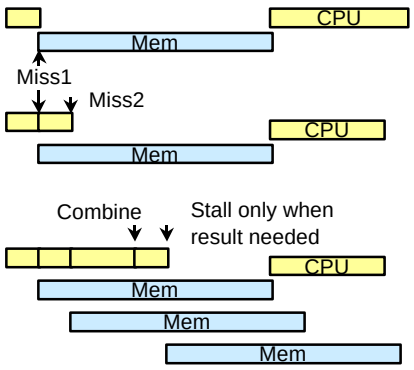
Victim Cache

- Conflict misses usually isolated to a *few* sets
- Use a small, fully-associative cache to handle these very popular sets
  - on eviction put the *victim* in the *victim cache*
  - Jouppi ISCA 1991

Set	1W	2W	3W	4W	
0	X				
1	X	X			
2	X				
3	X	X	X	X	X
4	X				
5	X				
6					
7	X	X			

Non-Blocking Cache

- Throughput depends on continuing memory operations after a miss.
  - long main memory latency
  - hit under miss
  - multiple outstanding
- Need to keep status of pending misses
  - destination register, target block, word in block
  - combine misses to same block
  - two options
    - keep status in cache block
    - keep in *miss status registers (MSRs or MSHRs)*



ILP Caches

High ILP implementations want to do more than one load/store per cycle.