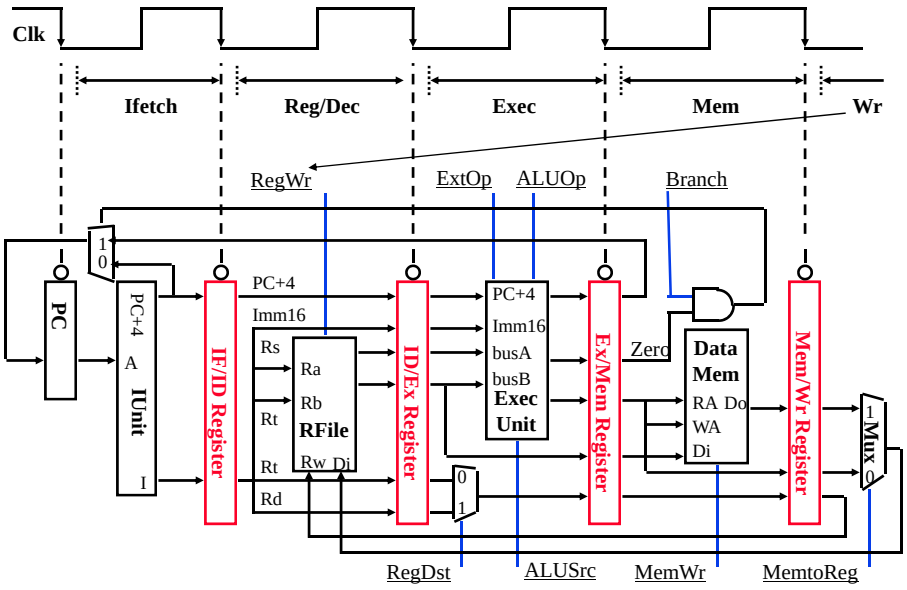
CS420/520
Computer Architecture I

Memory Hierarchy – Cache Systems

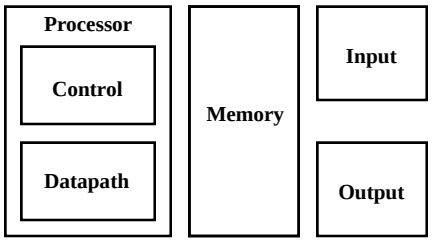
Dr. Xiaobo Zhou
Department of Computer Science

Review: A Pipelined Datapath



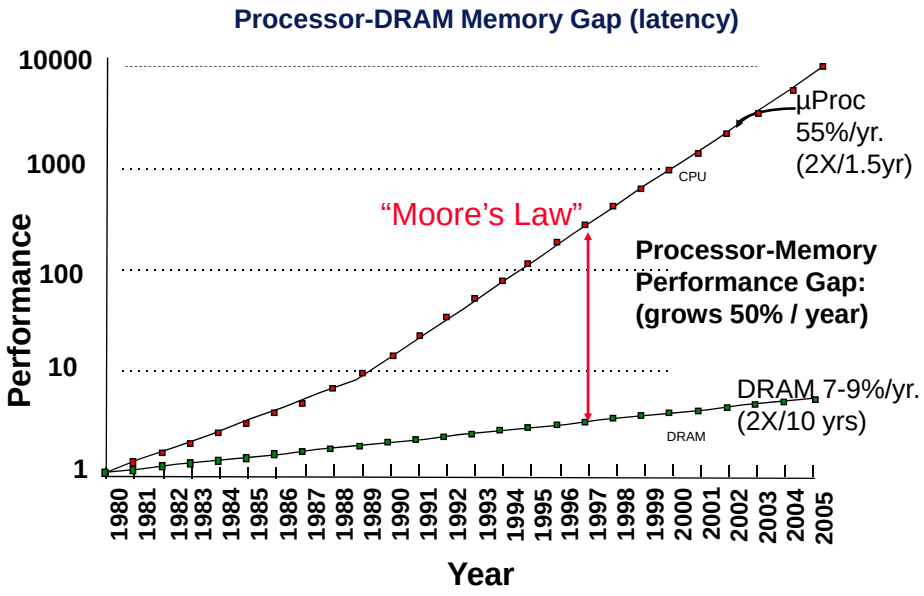
The Big Picture: Where are We Now?

° The Five Classic Components of a Computer



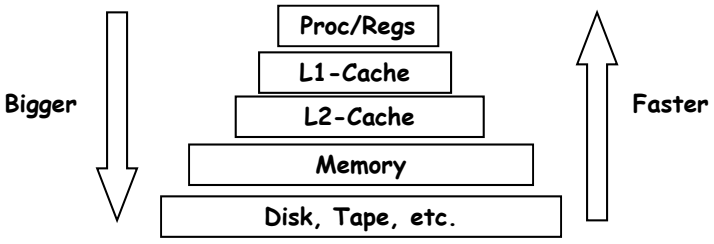
° Today's Topic: Memory System

Who Cares About the Memory Hierarchy?

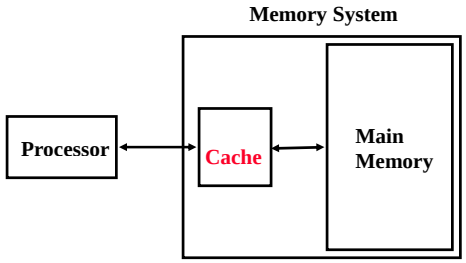


What is a Cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spatial and temporal locality
- In computer architecture, almost everything is a cache!
  - Registers a cache on variables
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)



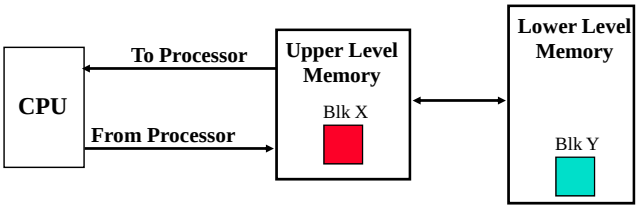
The Motivation for Caches



- Motivation:
  - Large main memories (DRAM) are slow
  - Small cache memories (SRAM) are fast
- Make the *average access time* small by:
  - Servicing most accesses from a small, fast memory
  - An Example
- Reduce the *bandwidth* required of the large main memory

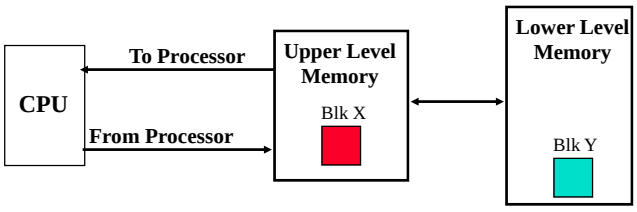
Memory Hierarchy: Principles of Operation

- At any given time, data is copied between only 2 adjacent levels:
  - Upper Level: the one closer to the processor
    - Smaller, faster, and uses more expensive technology
  - Lower Level: the one further away from the processor
    - Bigger, slower, and uses less expensive technology
- Block:
  - The minimum unit of information that can either be present or not present in the two level hierarchy



Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of
    Time to determine hit/miss + SRAM access time
- Miss: data needs to be retrieve from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level from lower level + Time to deliver the block to the processor
- Hit Time << Miss Penalty

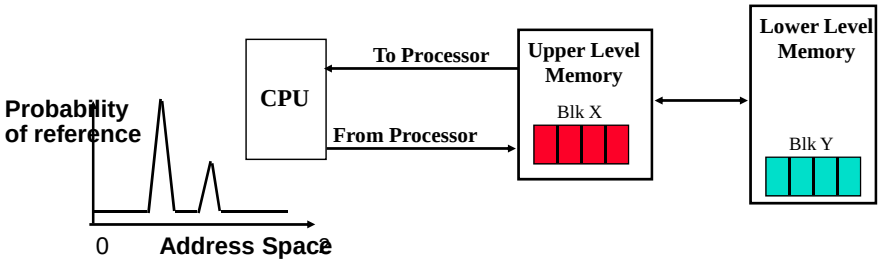


Memory Hierarchy: How Can it Work?

° **Principle of Locality** states the programs access a small portion of address space at any instant of time
  - Example: 90% of time in 10% of the code

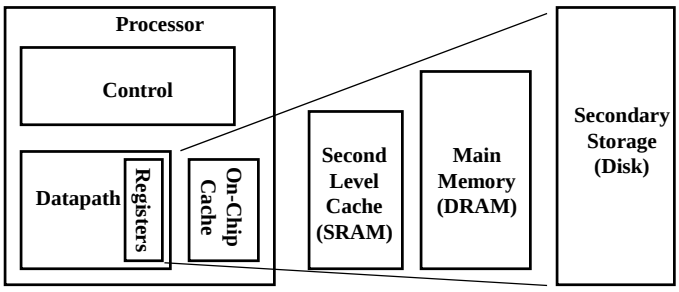
° **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon **(loops)**.
  - Keep more recently accessed data items closer to the processor

° **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon **(arrays)**.
  - Move blocks consists of contiguous words to the upper levels



Memory Hierarchy of a Modern Computer System

° By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.

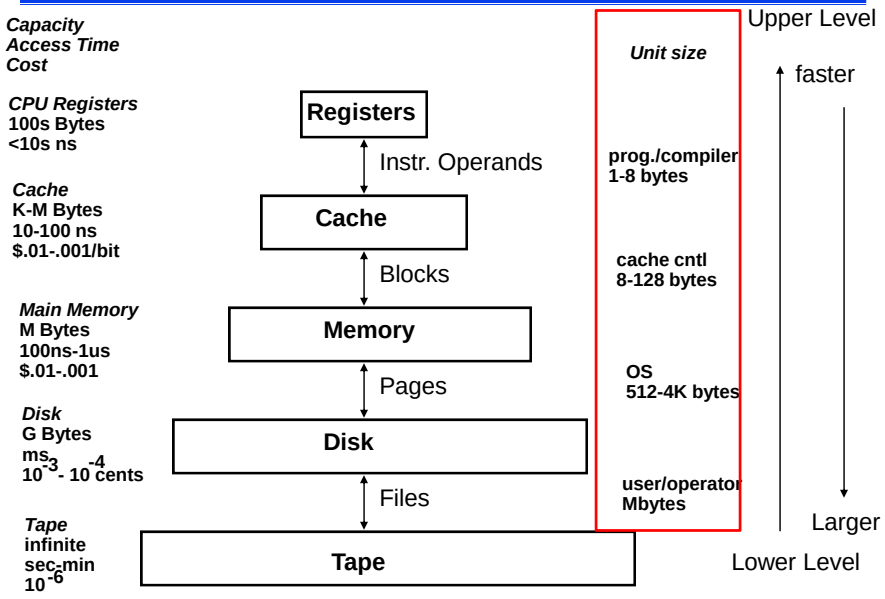


Speed (ns):	0.25ns	0.5ns – 25ns	50-250 ns	10,000,000 ns (5-10s ms)
Size (bytes):	100s	Ks	Ms	Gs

Memory Hierarchy Technology

- Random Access:
  - "Random" is good: access time is the same for all locations
  - DRAM: Dynamic Random Access Memory
    - High density, low power, cheap, slow
    - Dynamic: need to be "refreshed" regularly
  - SRAM: Static Random Access Memory
    - Low density, high power, expensive, fast
    - Static: content will last "forever"
- "Non-so-random" Access Technology:
  - Access time varies from location to location and from time to time
  - Examples: Disk, tape drive, CDROM
- The next two lectures will concentrate on random access technology
  - The Main Memory: DRAMs
  - Caches: SRAMs

Levels of the Memory Hierarchy

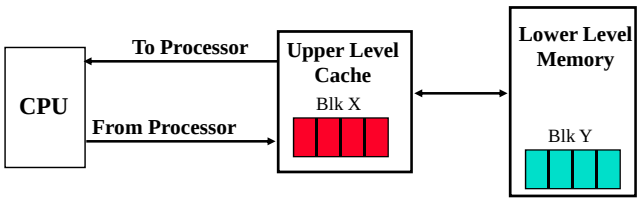


A Brief Summary

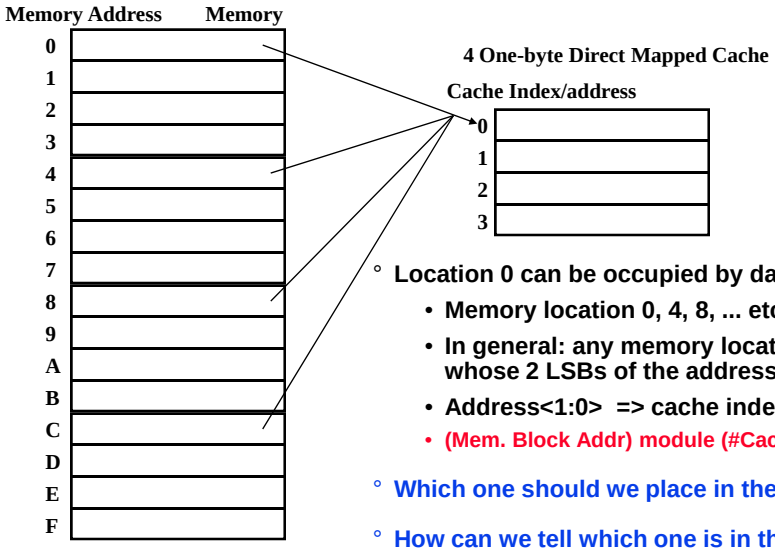
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology
  - Metric: average access time
- DRAM is slow but cheap and dense:
  - Good choice for presenting the user with a BIG memory system
- SRAM is fast but expensive and not very dense:
  - Good choice for providing the user FAST access time.

How Does Cache Work?

- Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Keep more recently accessed data items closer to the processor
- Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
  - Move blocks consists of contiguous data items to the cache
- What are issues/problems?
  - Where can a block be placed in the upper level (block placement)
  - How is a block found if it is in the upper level (block identification)
  - Which block should be replaced on a miss? (block replacement)
  - What happens on a write? (write strategy)



The Simplest Cache: Direct Mapped Cache



- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index
  - (Mem. Block Addr) module (#Cache blocks)
- Which one should we place in the cache?
- How can we tell which one is in the cache?

Example:

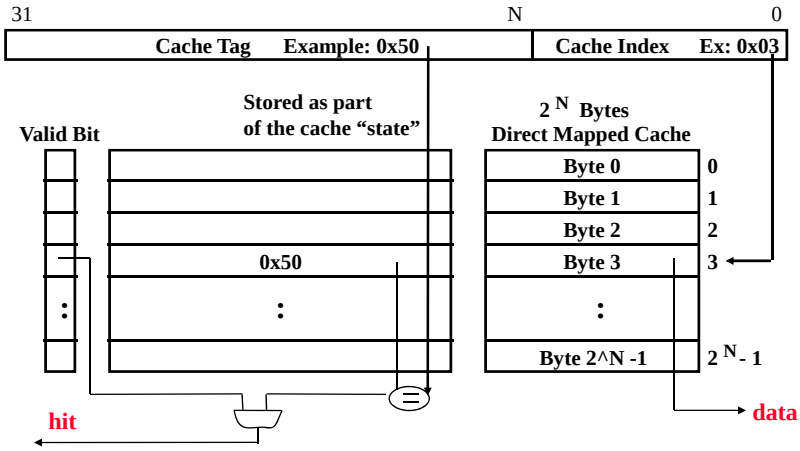
Consider an eight-word (block size is one word) direct mapped cache. Show the contents of the cache as it responds to a series of requests (decimal word addresses):

22, 26, 22, 26, 16, 3, 16, 18

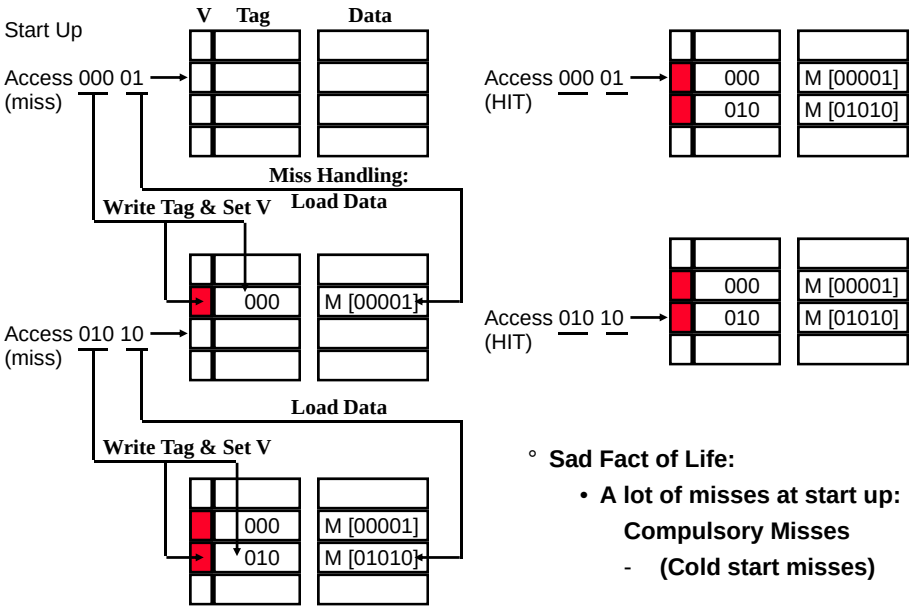
22	26	22	26	16	3	16	18
10110	11010	10110	11010	10000	00011	10000	10010
miss	miss	hit	hit	miss	miss	hit	miss
110	010	110	010	000	011	000	010

Cache Organization: Cache Tag and Cache Index

° Assume a 32-bit memory (byte) address:
- A 2^N bytes direct mapped cache:
  - Cache Index: The lower N bits of the memory address
  - Cache Tag: The upper (32 - N) bits of the memory address



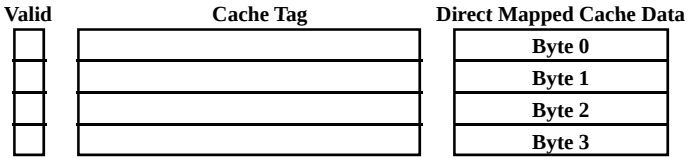
Cache Access Example



° Sad Fact of Life:
- A lot of misses at start up:
  Compulsory Misses
  - (Cold start misses)

Definition of a Cache Block

- Cache Block: the cache data that has in its own cache tag
- Our previous "extreme" example:
  - 4-byte Direct Mapped cache: Block Size = 1 Byte
  - Take advantage of Temporal Locality: If a byte is referenced, it will tend to be referenced soon.
  - Did not take advantage of Spatial Locality: If a byte is referenced, its adjacent bytes will be referenced soon.
- In order to take advantage of Spatial Locality: increase the block size



Example:

Consider a cache with 4 Bytes. Show the contents and the number of hits of the cache as it responds to a series of requests (decimal byte addresses): 0, 1, 4, 5… when block size is 1 byte and 2 bytes, respectively.

Mapping an Address to a Multiword Cache Block

Consider a cache with 64 Blocks and a block size of 16 bytes.
What cache block address does memory byte address 1200 map to?

Step 1: memory (byte) address → memory block address

How? memory (unit) address / # of units in a block (block size)

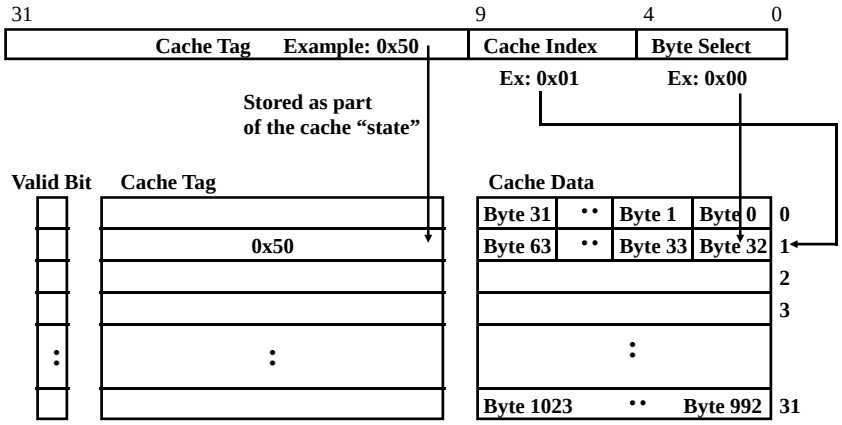
Step 2: memory block address → cache block address

How? DMC mapping scheme

Answer: (1200/16) module 64 = 11

Example: 1 KB Direct Mapped Cache with 32 B Blocks

° For a 2 ^ N byte cache:
- The uppermost (32 - N) bits are always the Cache Tag
- The lowest M bits are the Byte Select (Block Size = 2 ^ M)



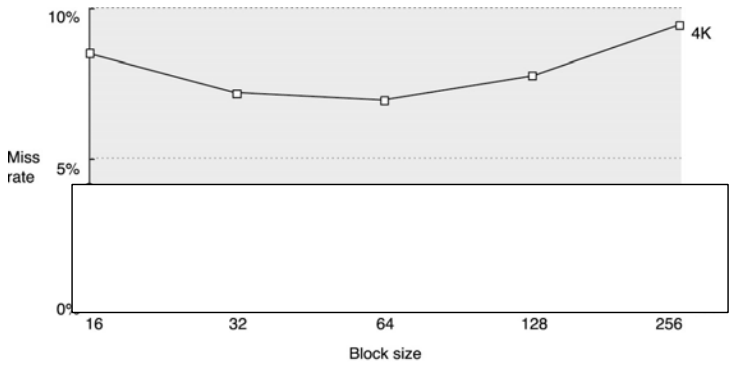
Total Bits in a Direct Mapped Cache

How many total bits are required for a directed mapped cache with 64 KB of data and one-word blocks, assuming a 32-bit address?

2^14 (32 + (32-14-2)+1) = 2^14*49 = 784 Kbits

2) How about 4-word blocks?

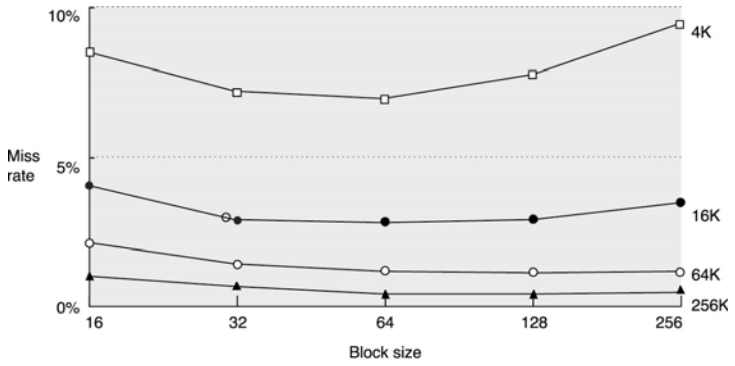
Miss Rate Reduction Technique I: Larger Block Size



Miss rate versus block size for four different-sized cache

° **Why the miss rate actually goes up if the block size is too large relative to the cache size?**

Miss Rate Reduction Technique II: Larger Caches



Miss rate versus block size for four different-sized cache

What is the price if the cache size goes up?

Average Memory Access Time

	Cache size			
Block size	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Figure 5.17 Actual miss rate versus block size for five different-sized caches in Figure 5.16. Note that for a 4 KB cache, 256-byte blocks have a higher miss rate than 32-byte blocks. In this example, the cache would have to be 256 KB in order for a 256-byte block to decrease misses.

Assume the main memory system takes 80 clock cycles of overhead per each access and then delivers 16 bytes every 2 clock cycles. Thus, it can supply 16 bytes in 82 cycles, 32 bytes in 84 cycles. Which block size has the smallest average memory access time for each cache size in the figure above? Assuming hit time 1 clock cycle, independent of block size.

Average Memory Access Time (II)

Block size	Miss penalty	Cache size 4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Figure 5.18 Average memory access time versus block size for five different-sized caches in Figure 5.16. Block sizes of 32 and 64 bytes dominate. The smallest average time per cache size is boldfaced.

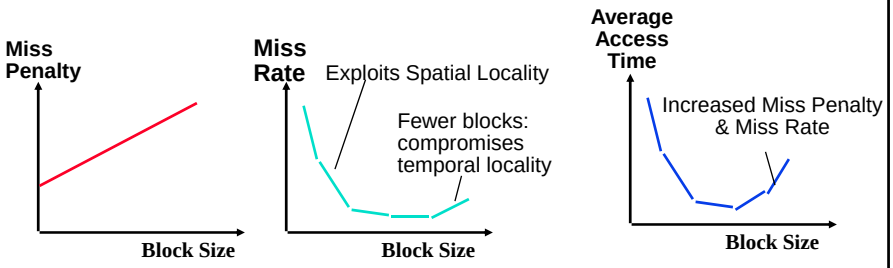
average memory access time = hit time + miss rate x miss penalty_@mem *

4K cache, block size 16 bytes: 1 + 8.57% x 82 = 8.027

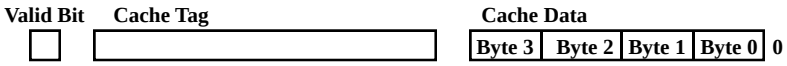
4K cache, block size 32 bytes: 1 + 7.24% x 84 = 7.082

Block Size Tradeoff

- **In general, larger block size take advantage of spatial locality BUT:**
  - **Larger block size means larger miss penalty:**
    - **Takes longer time to fill up the block**
  - **If block size is too big relative to cache size, miss rate will go up (as the room of taking temporal locality is reduced)**
- **Average Access Time:**
  - **= Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate**



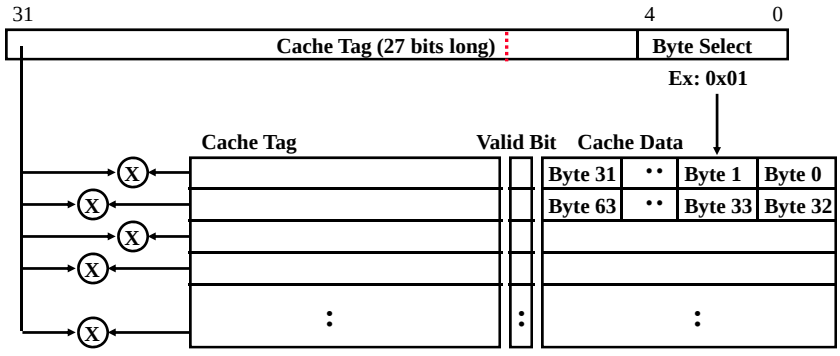
An Extreme Example



- Cache Size = 4 bytes Block Size = 4 bytes
  - Only ONE entry in the cache
- True: If an item is accessed, likely that it will be accessed again soon
  - But it is unlikely that it will be accessed again immediately!!!
  - The next access will likely to be a miss again
    - Continually loading data into the cache but discard (force out) them before they are used again
    - Worst nightmare of a cache designer: Ping Pong Effect
- Conflict Misses are misses caused by:
  - Different memory locations mapped to the same cache index
    - Solution 1: make the cache size bigger
    - Solution 2: Multiple entries for the same Cache Index

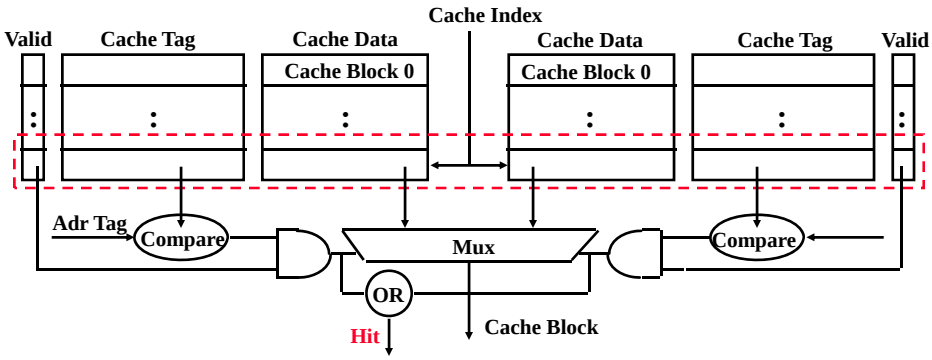
And yet Another Extreme Example: Fully Associative

- Fully Associative Cache
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32 bytes, we need N 27-bit comparators
- By definition: Conflict Miss = 0 for a fully associative cache



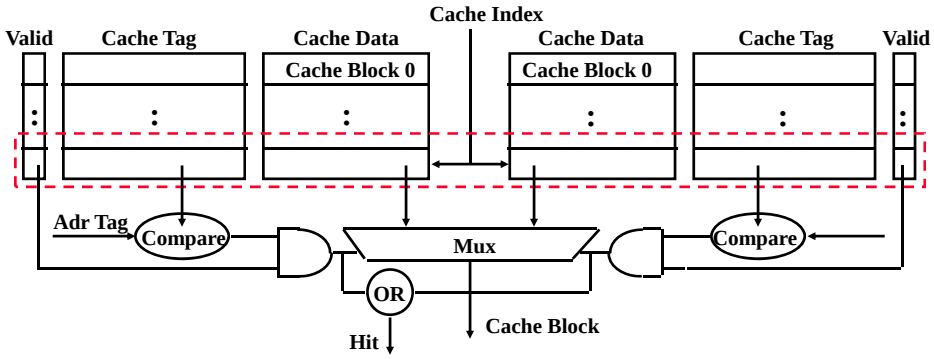
N-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - Cache Index selects a "set" from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



Disadvantage of Set Associative Cache

- N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



Miss Rate Reduction Technique III: Higher Associativity

Example:

There are three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set associative, and the third is direct mapped. Find the number of misses for each cache organization given the following sequence of word addresses: 0, 8, 0, 6, 8

* Unified mapping schemes:
  (Memory block address) module (# cache sets)

What if change the block size to 2-words?

A Common Framework for Memory Hierarchies

° **Question 1: Where can a block be placed?**

Scheme name	Number of Sets	Block per set
Directed mapped cache	Number of blocks in cache	1
Set Associative	$\frac{\text{Number of blocks in cache}}{\text{Associativity}}$	Associativity
Fully associative	1	# of blocks in cache

° **Question 2: How is a block Found?**

Associativity	Location method	Comparisons required
Directed mapped	index	1
Set associative	index the set	degree of associativity
Full	search all cache entries	size of the cache

A Summary on Sources of Cache Misses

- Compulsory (cold start, first reference):
  - first access to a *memory block*
    - Compulsory miss rate fixed once the block size fixed
  - "Cold" fact of life: not a whole lot you can do about it
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- Capacity:
  - Cache cannot contain all blocks access by the program
    - More than N distinct blocks used from last access -> capacity
  - Solution: increase cache size
- Invalidation: other process (e.g., I/O) updates memory

Miss Classification Example

- 4B cache, 1B blocks, directed mapped cache
  - 4 blocks, 4 sets

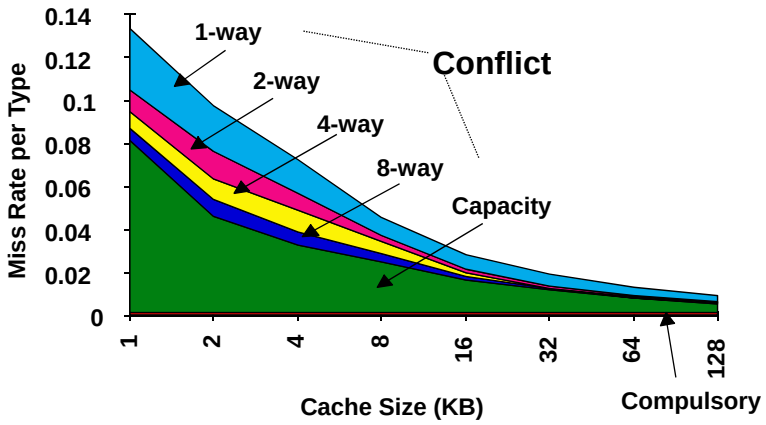
Current contents	Access address	Hit/Miss (which)
4, 1, 2, 7	4	hit
4, 1, 2, 7	8	Compulsory miss
8, 1, 2, 7	5	Compulsory miss
8, 5, 2, 7	1	Conflict miss
8, 1, 2, 7	6	Compulsory miss
8, 1, 6, 7	4	Capacity miss

First time -> Compulsory miss

N=4 distinct blocks used from the last access -> Capacity miss

Everything else -> Conflict miss

3Cs Absolute Miss Rate



- Compulsory misses (and rate) are independent of cache size (w/ fixed block size)
- Compulsory miss rate is very small
- Capacity misses decrease as capacity increase
- Conflict misses decreases as associativity increase

Q3: The Need to Make a Decision!

° **Direct Mapped Cache:**
  - **Each memory location can only mapped to 1 cache location**
  - **No need to make any decision :-)**
    - **Current item replaced the previous item in that cache location**

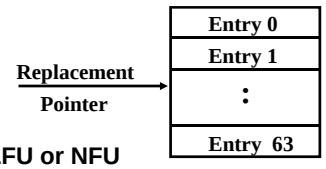
° **N-way Set Associative Cache:**
  - **Each memory location have a choice of N cache locations**

° **Fully Associative Cache:**
  - **Each memory location can be placed in ANY cache location**

° **Cache miss in a N-way Set Associative or Fully Associative Cache:**
  - **Bring in new block from memory**
  - **Throw out a cache block to make room for the new block**
  - **==> We need to make a decision on which block to throw out!**

Cache Block Replacement Policy

- Random Replacement:
  - Hardware randomly selects a cache item and throw it out
- Least Recently Used (LRU):
  - *Special hardware* keeps track of the access history
  - Replace the entry that has not been used for the longest time
- Example of a Simple "Pseudo"-LRU Implementation:
  - Assume 64 Fully Associative Entries
  - A replacement pointer points to one cache entry
  - Whenever an access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer
  - Example: 0, 5, 2, 0, 1, 5…
- FIFO Replacement
- NRU (Not Recently Used) Replacement or LFU or NFU



Least Recently Used (LRU)

- Must keep a linked list of blocks(pages)
  - most recently used at front, least at rear
  - update this list every memory reference !!
  - There are other simulation alternatives, but all costly
- Alternatively keep a counter in each block entry (in VM, in each page entry)
  - choose block(page) with lowest value counter
  - periodically zero the counter (NRU)
  - And more simulation alternatives

Re: Page Replacement Algorithms (Continued)

Not Recently Used (NRU):

Associated with each page is a reference flag (use/reference bit) such that reference flag = 1 if the block (in VM, page) has been referenced in recent past (read or write)
= 0 otherwise

-- if replacement is necessary, choose any block such that its reference bit is 0. This is a block that has not been referenced in the recent past period

In practice, a multi-class NRU considerers both *r* bit and *m* bit; see OS!

Data Cache Misses of LRU, Random, FIFO (Alpha)

Associativity						
Two-way				Four-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO
16KB	114.1	117.3	115.5	111.7	115.1	113.3
64KB	103.4	104.3	103.9	102.4	102.3	103.1
256KB	92.2	92.1	92.5	92.1	92.1	92.5

Per 1000 instructions; block size 64B; 10 SPEC2000

What it tells?

For the largest cache, little difference between different schemes.

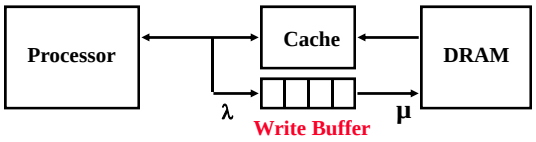
For smaller caches, LRU outperforms the others.

FIFO in general outperforms the random in the smaller caches.

Q4: Cache Write Policy: Write Through versus Write Back

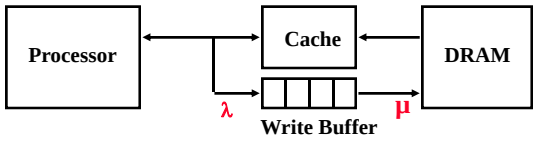
- Cache read is much easier to handle than cache write:
  - Instruction cache is much easier to design than data cache
- Cache write:
  - How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
  - Write Through: write to cache and memory at the same time.
    - Simplifies data coherency
    - What!!! How can this be? Isn't memory too slow for this?
  - Write Back: write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
    - Need a "dirty" bit for each cache block
    - Multiple writes within a block require one write to memory
    - Greatly reduce the memory bandwidth requirement
    - Control can be complex

Write Buffer for Write Through

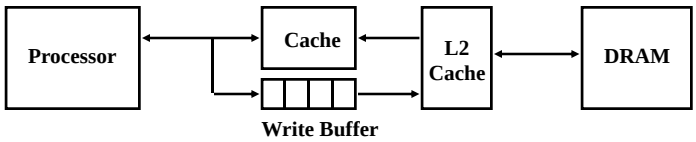


- A Write Buffer is needed between the Cache and Memory
  - It is for write through, not write back!
  - Processor: writes data into the cache and the write buffer (quick)
  - Memory controller: write contents of the buffer to memory (slow)
  - Purpose: overlap processor execution with memory updating
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (rate) << 1 / DRAM write cycle (DRAM write rate)

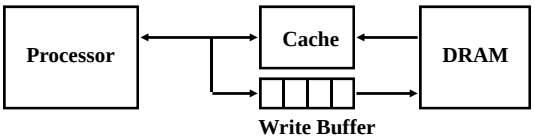
Write Buffer Saturation



- Store frequency (rate) λ → 1 / DRAM write cycle (DRAM write rate) μ
  - If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Store buffer will overflow no matter how big you make it; (limited-buffer queueing)
    - The CPU Cycle Time <= DRAM Write Cycle Time
- Solution for write buffer saturation:
  - Use a write back cache
  - Install a second level (L2) cache:



Write Buffer Consistency Issue



- Assume a direct-mapped, write-through cache that maps memory word address 512 and 1024 to the same cache block (index 0), and a four-word write buffer. Will the value in R2 always be equal to the value in R3?

```
SW R3, 512(R0)      ; M[512] ←R3     (cache index 0)
LW R1, 1024 (R0)    ; R1 ← M[1024]   (cache index 0)
LW R2, 512(R0)      ; R2 ← M[512]    (cache index 0)
```

- Write buffer complicates memory accesses because they might hold the updated value of a location needed on a read miss (RAW hazard)
  - Option1: Read miss to wait until the write buffer is empty
  - Option 2: check the contents of the write buffer on a read miss first!

    (Giving priority to Read Misses over Writes to reduce Miss Penalty)

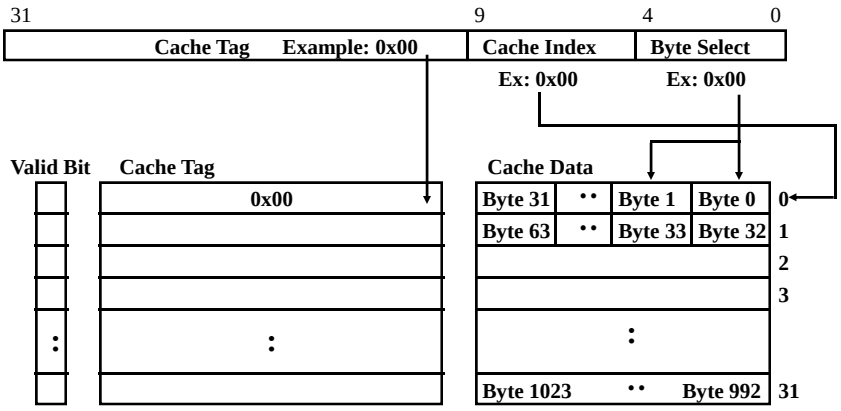
Write Allocate versus Not Allocate

° Assume: a 16-bit write to memory location 0x0 and causes a write miss

- Do we read in the rest of the block (Byte 2, 3, ... 31)?

  Yes: Write Allocate (act like read misses)

  No: Write Not Allocate (the block is modified only in the lower-level main memory)



Example:

Assuming a fully associative write-back cache with many cache entries that starts empty. Below is a sequence of five memory operations (the address is in square brackets).

Write Mem[100];
Write Mem[100];
Read Mem[200];
Write Mem[200];
Write Mem[100];

What are number of hits and misses when using write not allocation and write allocate, respectively?

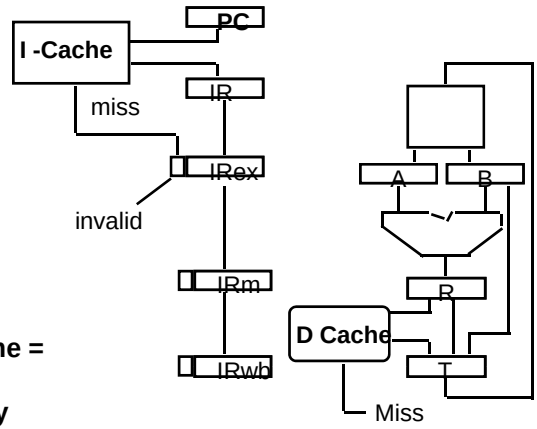
Write not allocate: m, m, m, h, m (4 misses and 1 hit)

Write allocate: m, h, m, h, h (2misses and 3 hits)

Observations: Write through + not allocate
Write back + allocate

Performance: Impact on Cycle Time

Cache Hit Time:
- directly tied to clock rate
- increases with cache size
- increases with associativity



Average Memory Access time =
(Hit Rate) x Hit Time +
Miss Rate x Miss Penalty

Time = IC x CT x (ideal CPI + average memory stalls)

Improving Cache Performance: 3 general options

Time = IC x CT x (ideal CPI + average memory stalls)

1. **Reduce the miss rate / increase the hit rate,**
2. **Reduce the miss penalty, or**
3. **Reduce the time to hit in the cache.**

Memory stall cycles = Read stall cycles + write-stall cycles

Read stall cycles = #reads * read miss rate * read miss penalty
= #i-reads * i-read miss rate * i-read miss penalty +
#d-reads * d-read miss rate * d-read miss penalty

Write stall cycles = #writes * write miss rate * write miss penalty

Memory stall cycles = #memory access * average miss rate * miss penalty

$$= IC * \frac{\text{memory access}}{\text{Instruction}} * \text{average miss rate} * \text{miss penalty}$$

Memory Performance Example I

Q1: Assume an instruction cache miss rate for gcc of 2% and a data cache miss rate of 4%. If a machine (M1) has a CPI of 2 if without any memory stalls and the miss penalty is 40 cycles for all misses, determine how much faster a machine (M2) would run with a perfect cache that never missed. It is known that the frequency of all loads and stores in gcc is 36%

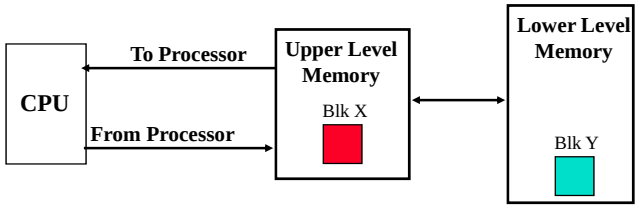
Answer:

Instruction miss cycles = I * 2% * 40 = 0.80I
Data miss cycles = I * 36% * 4% * 40 = 0.576I = 0.58 I
M1: The CPI with memory stalls is 2 + 1.38 = 3.38
M2: The CPI with a perfect cache is 2
The performance with the perfect cache is:
Per_M2/Per_M1 = 3.38 / 2 = 1.69

Q2: Suppose we speed up the machine by reducing CPI from 2 to 1 without changing the clock rate & memory system, how much faster M2 than M1

Memory Performance Example I (cont.):

Q3: If we double clock rate (M3), assuming the absolute time (miss penalty) to handle a cache miss does not change, how much faster M3 than M1 (Q1)



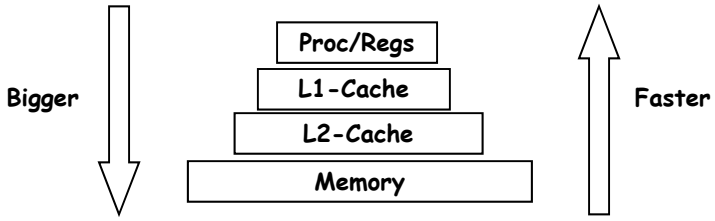
Miss penalty in terms of # clock cycles doubles!

Answer:

Instruction miss cycles = I * 2% * 80 = 1.60I
Data miss cycles = I * 36% * 4% * 80 = 0.576I = 1.16 I
M3: The CPI with memory stalls is 2 + 2.76 = 4.76
M1: The CPI is 3.38
The performance with the fast clock is:
Per_M3/Per_M2 = 3.38 / (4.76 x ½) = 1.42

Multi-level Cache to reduce Miss Penalty

- Should we make the cache faster to keep pace with the speed of CPUs, or make the cache larger to overcome the widening gap between the CPU and main memory?
  - If both, multi-level caches
  - In a two-level modern cache system
    - the first-level small enough to match the clock cycle time of fast CPU
    - The second-level cache can be large enough to capture many access that would go to slow main memory so as to reduce the effective miss penalty – second level faster than MM

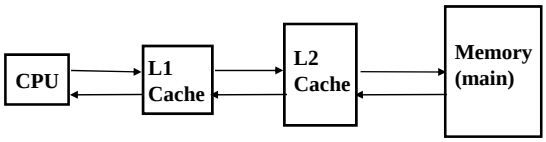


Local Miss Rate and Global Miss Rate

- Local miss rate: the number of misses in a cache divided by the total number of memory accesses to this cache
  - Miss rate_L1: the number of misses in a cache divided by the total number of memory accesses from CPU to L1 cache
  - Miss rate_L2: the number of misses in a cache divided by the total number of memory accesses from L1 cache to L2 cache
- Global miss rate: the number of misses in the cache divided by the total number of memory access generated by the CPU
  - Level one: global miss rate = local miss rate
  - Level two: global miss rate = Miss rate_L1 x Miss rate_L2
- Average memory access time = Hit time_L1 + Miss rate x Miss pentaly_L1

  Miss pentaly_L1 = Hit time_L2 + Miss rate_L2 x Miss pentaly_L2

  << Miss pentaly_L2



Performance of A Two-level Cache

- Suppose that in 1000 memory accesses, 40 misses in the first-level cache and 20 misses in the second-level cache.

  Question 1: what are the various miss rates?

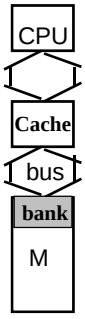
  Miss rate_L1 = 40/1000 = 4% for both local and global
  Local Miss rate_L2 = 20/40 = 50%
  Global Miss rate_L2 = 20/1000 = 2% (or 4% * 50%)

- Question 2: assume the miss penalty from L2 cache to memory is 100 cycles, hit time of L2 cache is 10 cycles, the hit time of L1 cache is 1 cycle. What is the average memory access time?

  Average memory access time = Hit time_L1 + Miss rate_L1 * (hit time_L2 + miss rate_L2 * miss penalty_L2)
  = 1 + 4% * (10 + 50% * 100) = 1 + 4% * 60 = 3.4 clock cycles

  or Average memory access time = (1000 * 1 + 20 * 10 + (40 - 20) * (100+10)) / 1000 = (1000 + 200 + 2200) / 1000 = 3.4 clock cycles

Main Memory Organizations to reduce Miss Penalty

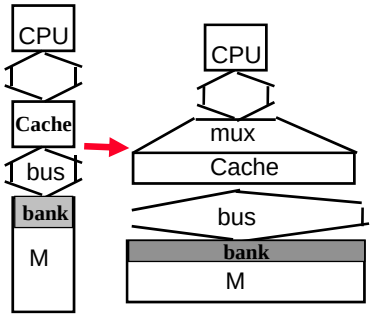


Basic Memory organization (A)
One-word-wide bank & bus

- First-level caches are often organized with a physical width of 1 word because most CPU access are that size.
- Performance factors:
  - 4 clock cycles to send a bank (a word) address
  - 56 clock cycles for the access time per bank (word)
  - 4 clock cycles to send a bank (word) of data via bus
- Given a cache block of 4 words, performance/Miss Penalty of a one-word-wide memory is
  - 4 * 4 + 4 * 56 + 4 * 4 = 256 cycles

Memory Transfer Time =
Memory access time + Bus transfer time
Memory access time >> Bus transfer time

First Technique: Wider Main Memory

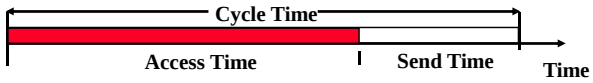


Wide Memory organization (B)
Wide Path Between Memory & Cache
(Higher cost in Bus & Memory)

- Performance factors:
  - 4 clock cycles to send a memory bank address
  - 56 clock cycles for the access time per memory bank
  - 4 clock cycles to send a memory bank of data via bus
- Given a cache block of 4 words, performance/Miss Penalty of a two-word-wide memory is
  - 2 * 4 + 2 * 56 + 2 * 4 = 128 cycles
- Given a cache block of 4 words, performance/Miss Penalty of a four-word-wide memory is
  - 4 + 56 + 4 = 64 cycles

Some difference between CA and CO in address sending overhead!

Cycle Time versus Access Time



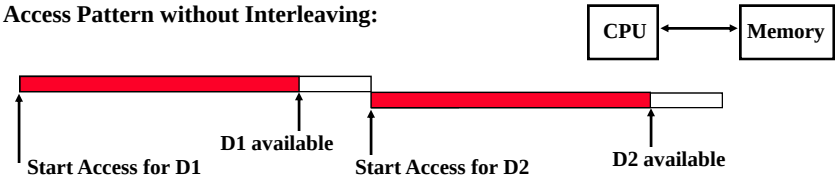
- DRAM (Read/Write) Cycle Time >> DRAM (Read/Write) Access Time

How about to reduce access time by increasing memory bandwidth?

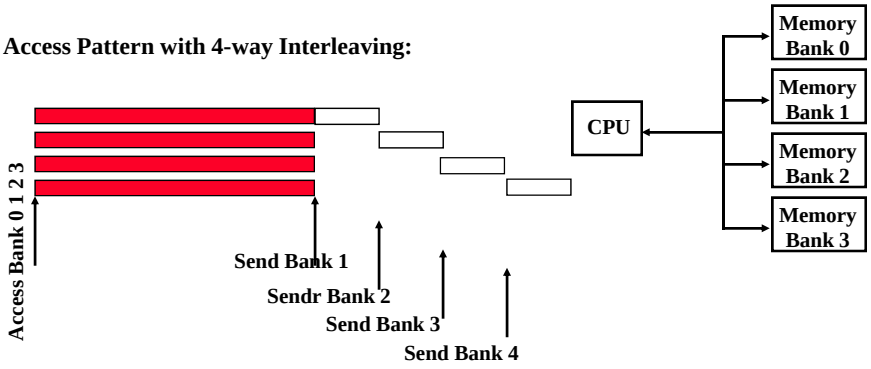
$$\text{\# of Memory access} = \frac{\text{Cache Block Size}}{\text{\# of banks x bank size}}$$

Increasing Memory Bandwidth - Interleaving

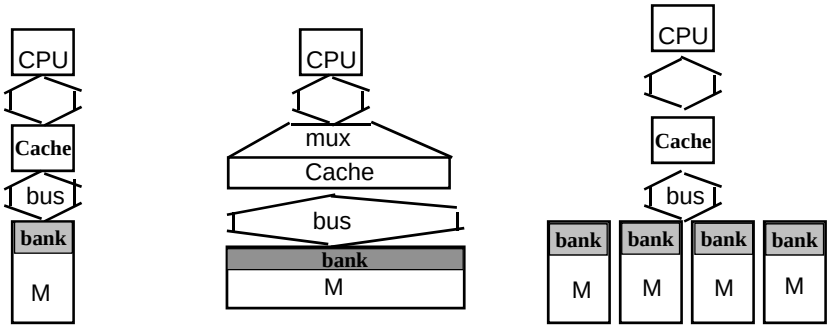
Access Pattern without Interleaving:



Access Pattern with 4-way Interleaving:



Second Technique: Interleaved Memory



Memory organization A
One-word-wide bank & bus

Wide Memory organization (B)
Wide Path Between Memory & Cache

Memory organization (C)
Memory Interleaving

- **Given a cache block of 4 words, performance/Miss Penalty of a four-way interleaving memory (4 one-word banks) is**
  - **4 + 56 + 4 x 4 = 76 cycles**
  - **Or 4 x 4 + 56 + 4 x 4 = 88 cycles**

Main Memory Performance Example

- Timing model
  - 1 clock cycle to send bank address,
  - 6 access time for one bank, 1 to send one-bank data
  - Cache Block is 4 words
- *Simple M.P.*
  - No interleaving, one-word-bank, one-word bus
    - = 4 + 4 x 6 + 4 x 1 = 32
- *Wide M.P.*
  - Four-word bank, four-word bus: 1 + 6 + 1 = 8
  - Two-word bank, two-word bus: 2 + 2 x 6 + 2 x 1 = 16
- *Interleaved M.P.*
  - 4-way interleaving memory, one-word-bank, one-word bus: 4 + 6 + 4 x 1 = 14
  - 2-way interleaving memory, two-word-bank, two-word bus: 2 + 6 + 2 x 1 = 10

Common Framework for Memory Hierarchy

- Question 1: Where can a Block be Placed
  - Cache:
    - direct mapped, n-way set associative
  - VM:
    - fully associative
- Question 2: How is a block found
  - index,
  - index the set and search among elements
  - search all cache entries or separate lookup table
- Question 3: Which block be replaced
  - Random, LRU, NRU (not-recently-used), FIFO
- What happens on a write
  - write through vs write back
  - write allocate vs write no-allocate on a write miss

Summary of Six Basic Cache Optimization

- Larger block size to reduce miss rate
- Bigger cache to reduce miss rate
- Higher associativity to reduce miss rate (conflict miss)
- Multi-level caches to reduce miss penalty
- Giving priority to read misses over writes to reduce miss penalty
- Avoiding address translation during cache indexing to reduce hit time

Eleven Advanced Cache Optimizations

- Reducing the hit time: small and simple caches, way prediction, and trace caches
- Increasing cache bandwidth: pipelined caches, multi-banked caches, and non-blocking caches
- Reducing the miss penalty: critical word first and merging write buffers
- Reducing the miss rate: compiler optimization
- Reducing the miss penalty or miss rat via parallelism: hardware pre-fetching and compiler pre-fetching
- Where to read for memory hierarchy?
  - CO 4: Chapter 5
  - CA 5: Appendix B

Summary:

- The Principle of Locality: Temporal Locality vs Spatial Locality
- Four Questions For Any Cache
  - Where to place in the cache
  - How to locate a block in the cache
  - Replacement: Random, LRU, NRU, LFU
  - Write policy: Write through vs Write back
    - Write miss: Write Allocate vs. Write Not Allocate
    - Write buffer
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size
- Three general options to improve cache performance
  - Reduce the miss rate / increase the hit rate,
  - Reduce the miss penalty, or
  - Reduce the time to hit in the cache