Memory Hierarchy

Dr. Arjan Durresi
Louisiana State University
Baton Rouge, LA 70810
Durresi@Csc.LSU.Edu

These slides are available at:

http://www.csc.lsu.edu/~durresi/CSC7080_05/

Overview

- Basics of Caches
- Measuring and Improving Cache Performance
- Framework for Memory Hierarchies

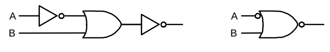
Memory

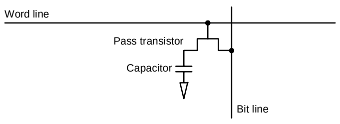
- "You can never be too rich, too thin, or have too much memory“
- Create the illusion of unlimited fast memory
- Analogy with the use of library books
  - Get the books you need once
  - Or go to library each time you need new data
- Temporal Locality: if an item is referenced, it will tend to be referenced again soon.
- Spatial Locality: If an item is referenced, items whose addresses are close by will tend to be referenced soon
- Locality comes from natural program structure such as loops, arrays etc.
- Memory hierarchy: multiple levels of memory with different speeds and sizes

---

Memories: Review

- SRAM:
  - value is stored on a pair of inverting gates
  - very fast but takes up more space than DRAM (4 to 6 transistors)
- DRAM:
  - value is stored as a charge on capacitor (must be refreshed)
  - very small but slower than SRAM (factor of 5 to 10)





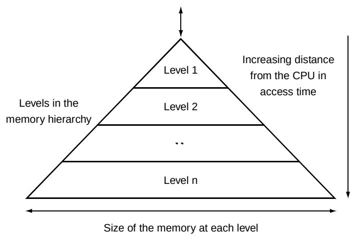
Exploiting Memory Hierarchy

- Users want large and fast memories!

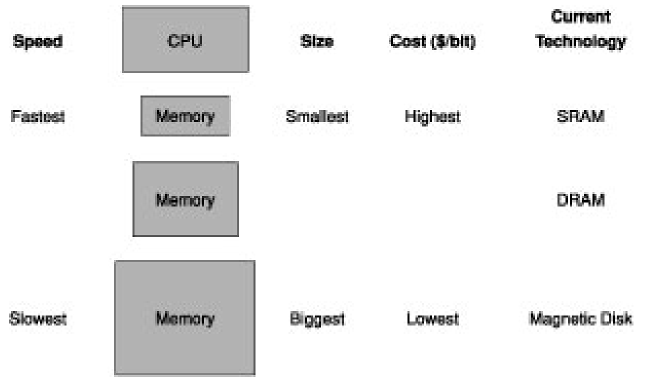
  SRAM access times are .5 – 5ns at cost of $4000 to $10,000 per GB.
  DRAM access times are 50-70ns at cost of $100 to $200 per GB.
  Disk access times are 5 to 20 million ns at cost of $.50 to $2 per GB.

  2004

- Try and give it to them anyway
  - build a memory hierarchy



Basic Structure of Memory Hierarchy



Speed	CPU	Size	Cost ($/bit)	Current Technology
Fastest	Memory	Smallest	Highest	SRAM
	Memory			DRAM
Slowest	Memory	Biggest	Lowest	Magnetic Disk

The user has the illusion of a memory as large as the largest level,
But can be accessed as if it were all built from the fastest one

Locality

- A principle that makes having a memory hierarchy a good idea
- If an item is referenced,

temporal locality: it will tend to be referenced again soon
spatial locality: nearby items will tend to be referenced soon.

Why does code have locality?

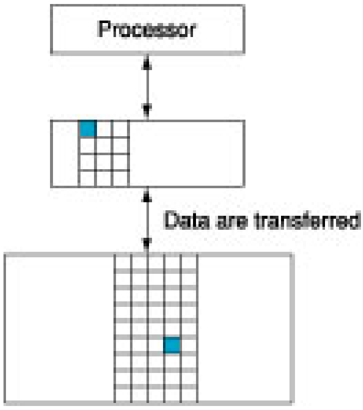
- Our initial focus: two levels (upper, lower)
  - block: minimum unit of data
  - hit: data requested is in the upper level
  - miss: data requested is not in the upper level

Memory Hierarchy

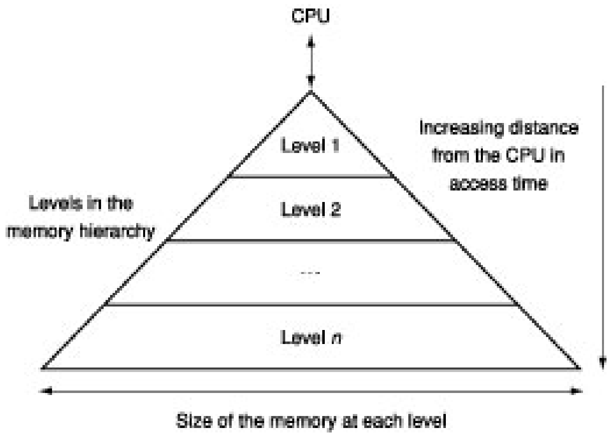
Miss rate: the fraction of memory
Access not found in a level of the
Memory hierarchy

Hit time: the time required to
Access a level of the hierarchy,
Including the time needed to determine
The access is a hit or a miss

Miss penalty: the time required to fetch
a block into a level of the memory
Hierarchy from the lower level, including
the time to access the block, transmit it, and
insert in the needed level



Memory Hierarchy



Cache

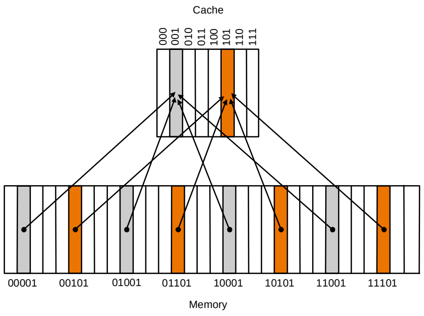
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - block size is one word of data
  - "direct mapped"

For each item of data at the lower level,
there is exactly one location in the cache where it might be.

e.g., lots of items at the lower level share locations in the upper level

Direct Mapped Cache

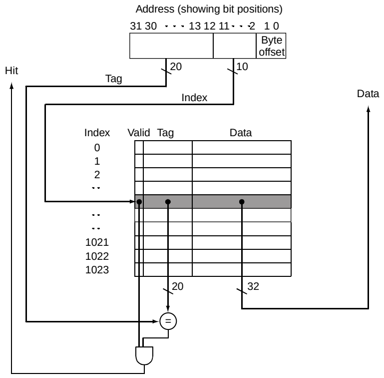
- Mapping: address is modulo the number of blocks in the cache



(Block Address) modulo (Number of cache blocks in the cache)

Direct Mapped Cache

- For MIPS:



Index - used to select the word
Tag - used to compare with the value of tag field of the cache
Valid bit – indicate whether an entry contains a valid address

Cache

- How many bits are required for a direct-mapped cache with 16 KB of data and 4-word blocks, assuming 32-bit address?
- 16 KB –> 4Kwords, 2^{12} words, with a block size of 4 words –> 2^{10} blocks
- Each block has 4 x 32 = 128 bits data plus a tag
- If number of blocks = 2^n, word = 2^m, the tag field = 32 – n – m -2
  - Tag = 32 - 10 - 2 -2 = 18
- Total cache = 2^{10} x (128 + (32 – 10 -2 -2) +1) = 2^{10} x 147 = 147 Kbits = 18.4 KB
- So 18.4 KB for a 16 KB cache, total number of bits is 1.15 times as nedded just for the storage of data

---

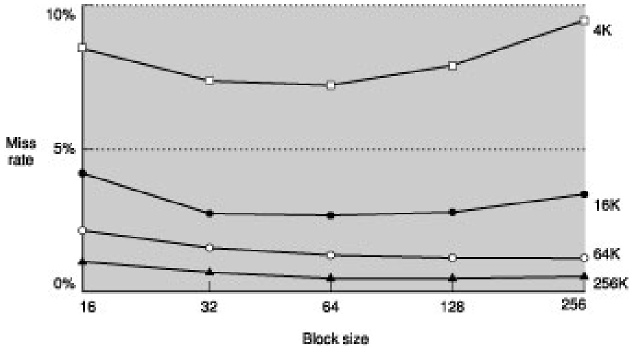
Mapping to Cache

- A cache with 64 blocks and a block size of 16 bytes.
- What block number does byte address 1200 map to?
- (Block Address) modulo (Number of cache blocks in the cache)
- The address of the block = $\frac{\text{Byte address}}{\text{Bytes per block}} = 75$
- That maps to cache block number (75 modulo 64) = 11
- This block maps all addresses between 1200 and 1215
- Larger blocks exploit spatial locality to lower miss rate
- But when block too big relatively to cache – more competition for those blocks - the block will be out of cache before many of its words are used

Block Size

- Increasing the block size – cost of a miss increases
- Miss penalty - time required to fetch the block from the next level of hierarchy and load it into the cache.
- Time to fetch
  - Latency to the first word
  - Transfer time for the rest of the block
- When the block size increases
  - The improvement in the miss rate starts to decrease
  - The increase in miss penalty > improvement in miss rate
  - The cache performance decreases

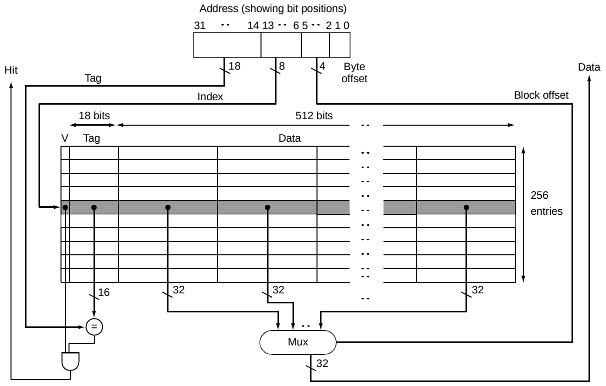
Miss Rate vs. Block Size



Each line represents a cache of different size
The miss rate goes up if the block size is too large relative to cache size.

Direct Mapped Cache

- Taking advantage of spatial locality:



For SPEC200 : Instruction miss rate = 0.4%, Data miss rate = 11.4%, Effective combined miss rate = 3.2%

Hits vs. Misses

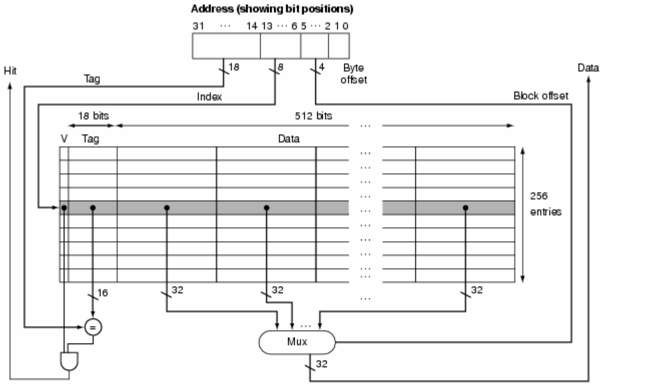
- Read hits
  - this is what we want!
- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart
  - Similar to pipeline stalls. What is the difference among them?
- Steps for cache (instruction) miss :
  - Send the original PC value (current PC -4) to the memory
  - Instruct main memory to perform a read and wait for the memory to complete access.
  - Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address from ALU into the tag field, and turning the valid bit on.
  - Restart the instruction execution at the first step, which will refetch the instruction, this time in the cache.

Writes

- Write hits:
  - Avoid inconsistency between cache and memory
  - can replace data in cache and memory (write-through)
    - Writes very slow – 100 processor clock
    - SPEC2000 integer – 10% of instructions are stores
    - If CPI without writes is 1.0, spending 100 extra cycles on every writes: 1.0 + 100x10% = 11
  - Use write in cache and in buffer – a write buffer stores the data while it is waiting to be written into memory.
    - If buffer is full – processor will stall
    - To avoid stalling – increase the depth of the buffer
  - write the data only into the cache (write-back the cache later). It written into memory when replaced. More complex mechanism.

- Write misses:
  - read the entire block into the cache, then write the word
  - Write the word into memory
  - Write-back
    - Uses a store buffer. Two cycles – one to write in the buffer, second to write from the buffer to the cache.
  - Write-through – can be done in one cycle

FastMATH Example



FastMATH – a fast embedded microprocessor – MIPS architecture It has separate instruction and data cache. Each cache is 16KB, or 4K words, with 16-word blocks.

Read

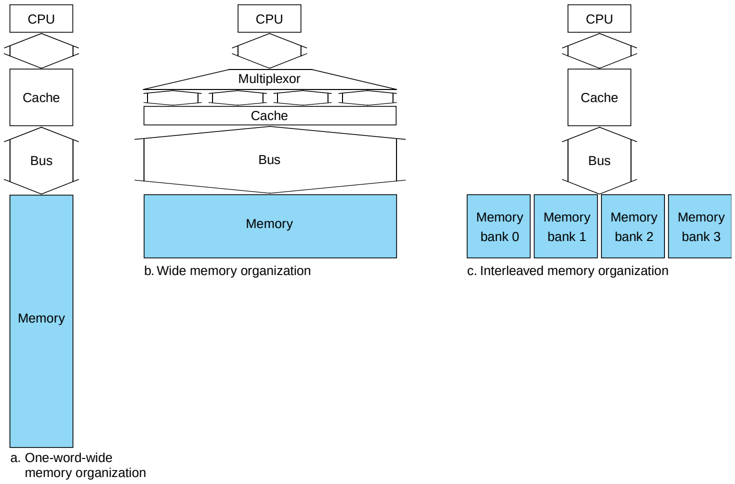
- Send address to the appropriate cache. The address comes from the PC (for an instruction) or from ALU (for data)
- If it is a hit, the requested word is available on the data lines. Since there are 16 words in the desired block, it is selected using a multiplexor.
- If it is a miss, the address is sent to the main memory. When the memory returns the data, it is written into the cache and then read.

Writes

- FastMATH offers both write-through and write-back
- Operating systems decide which strategy to use for a given application
- It has a one-entry write buffer
- Miss rate for SPEC2000 banchmarks:
  - Instruction miss rate = 0.4%
  - Data miss rate = 11.4%
  - Effective combined miss rate = 3.2% . Depends on the frequency of data and instructions rate.

Hardware Issues

- Make reading multiple words easier by using banks of memory



a. One-word-wide memory organization

b. Wide memory organization

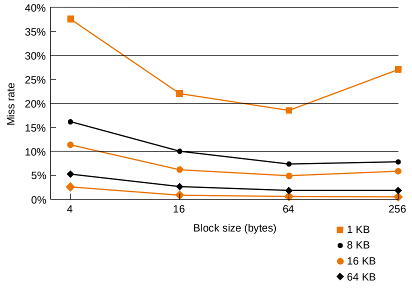
c. Interleaved memory organization

Access Time

- 1 memory bus clock cycle to send the address
- 15 memory bus clock for each DRAM access initiated
- 1 memory bus clock cycle to send a word data
- If we have a cache of four words:
- One-word-wide memory - Miss penalty = 1+ 4 x 15 + 4 x 1 = 65 memory bus clock cycles
  - Number of bytes transferred per bus clock for a single miss 4x4/65=.25
- With 2 word width (Fig b, slide 18) miss penalty = 1 + 2x15 + 2x1= 33
  - The bandwidth of a single miss is 0.48
- With 4 word width (Fig b) miss penalty = 1 + 15 + 1= 17
  - The bandwidth of a single miss is 0.94
- Try to have access initiated in parallel, use banks of memories (Fig c, slide 19) Interleaving scheme:
  - Miss penalty = 1 + 1x15 + 4x1 = 20 memory bus clock cycles
  - This is an effective bandwidth per miss of 0.80 bytes per clock, or about

Performance

- ❑ Increasing the block size tends to decrease miss rate:



Use split caches because there is more spatial locality in code:

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

Performance

- ❑ Simplified model:

execution time = (execution cycles + Memory-stall cycles) × cycle time

Memory-stall cycles = # of instructions × miss ratio × miss penalty

- ❑ Memory-stall= Read-stall cycles + Write-stall cycles
- ❑ Read-stall cycles = Reads/Program × Read miss rate × Read miss penalty
- ❑ Writes are more complex:
  - ❍ Write-through: write misses and buffer stalls (when buffer is fill)

  Write-stall cycles = Write/Program × Write miss rate × Write miss penalty + Write buffer stalls

  - ❍ With a deep buffer (e.g. four or more words) and a memory of accepting writes at a rate higher than the average program write frequency – write buffer stall can be ignored

Example of Cache Performance

- Assume an instruction miss rate for a program is 2% and a data cache miss rate is 4%
- If CPI is 2 without any memory stalls
- Miss penalty is 100 cycles for all misses
- How much faster it would run using SPECTint2000 without miss?
- The number of memory miss cycles for instructions in terms of instructions I is :

  Instruction miss cycle = I x 2% x 100 = 2.00 x I
- The frequency of stores in SPECint2000 is 36%:

  Data miss cycles = I x 36% x 4 % x 100 = 1.44 x I
- Total number of memory-stall cycles is 2 I + 1.44 I = 3.44 I
- CPI = 2 + 3.44 = 5.44
- Performance improvement = 5.44/2 = 2.72, amount of time in memory stalls = 3.44/5.44 = 63%
- If we speed up the processor CPI = 1, Performance improvement = 4.44, amount of time in memory stalls = 3.44/4.44 = 77%

Example (cont.)

- If we increase the processor clock rate by 2
- The miss penalty will be twice as many clocks – 200 clock cycles:

  Total miss per instruction = 2%x200 + 36%x4%x200 = 6.88
- The faster compute will have CPI = 2 + 6.88 = 8.88
- Let us compare the two computers:

$$\frac{\text{Performance fast clock}}{\text{Performance slow clock}} = \frac{\text{Execution time slow clock}}{\text{Execution time fast clock}}$$

$$= \frac{\text{IC x CPI}_{slow} \text{ x Clock cycles}}{\text{IC x CPI}_{fast} \text{ x } \frac{\text{Clock cycles}}{2}} = \frac{5.44}{8.88 /2} = 1.23$$

So the performance improvement is degraded fro 2 to 1.23 because cache misses

Example (cont.)

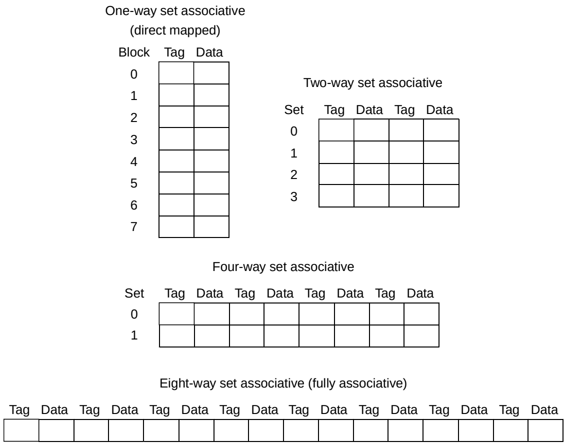
- If we improve both – lower CPI and higher clock
- The lower the CPI the more pronounced is the impact of stall cycles
- The higher the processor rate – larger miss penalty

Improving Performance

- Two ways of improving performance:
  - decreasing the miss ratio
  - decreasing the miss penalty

What happens if we increase block size?

Decreasing miss ratio with associativity



Position of a memory block

- In direct-mapped:
  - (Block number) modulo (Number of cache blocks)
- In set-associative, the set containing the memory block:
  - (Block number) modulo (Number of sets in the cache)

Misses ann Associatively in Caches

- Assume three caches, each 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 given the following sequence of addresses: 0, 8, 0, 6, 8
- Direct mapped:
  - Cache block = Block address modul 4
  - 0=0module4, 2=6 module 4, 0=8module 4

Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		0	1	2	3
0	miss	Mem[0]			
8	miss	Mem[8]			
0	miss	Mem[0]			
6	miss	Mem[0]		Mem[6]	
8	miss	Mem[8]		Mem[6]	

Misses and Associatively in Caches

- The set-associative has two sets (with indices 0 and 1) with two elements per set.
- Set = address module 2
- 0=0module2, 0=6module2, 0=8module2
- Replece the least recently used block within a set

Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		Set0	Set0	Set1	Set1
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	Mem[6]		
8	miss	Mem[8]	Mem[6]		

Misses and Associatively in Caches

- The fully associative has four blocks. Three misses
- If we had 8 blocks in cache, there would be no replacement s - three misses
- If we had 16 blocks – the three caches would have the same performance
- Cache size and associativity are related in determining cache performance

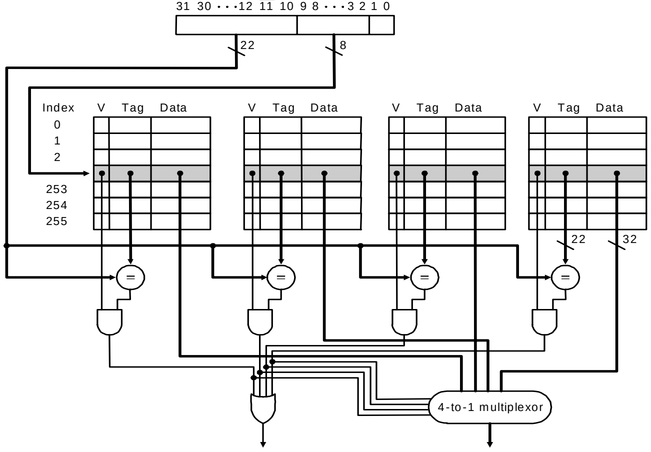
Address of memory Block accessed	Hit or miss	Contents of a cache blocks after reference			
		Block00	Block1	Block2	Block3
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	Mem[8]	Mem[6]	
8	hit	Mem[8]	Mem[8]	Mem[6]	

Performance

- SPEC2000 benchmarks for a 64KB data cache with a 16-word block. Associativity from one to eight way

Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

An implementation



Tags are searched in parallel

Each increase by two in associativity doubles the number of blocks per set and halves the number of sets – decreases the size of index by 1 bit and increase the size of tag by 1 bit

---

How much associativity ?

- The choice among direct-mapped, set-associative mapping will depend:
  - On the cost of a miss versus
  - The cost of implementing associativity

Size of Tags vs. Set Associativity

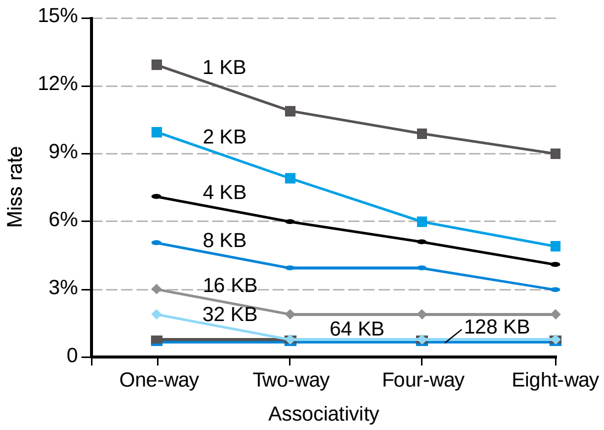
- Assuming a cache of 4K blocks, a four-word block size and a 32-bit address
- Find the total number of sets and total number of tag bits for caches
  - Direct-mapped
  - Two-way, four-way set associative and full associative
- There are 16 (2^4) bytes per block
- 32 bit address -> 32 - 4 = 28 bits to be used for index and tag
- Direct-mapped – same number of sets as blocks
  - $\log_2(4K)$ = 12 bits of index, total number of tag bits (28-12)x4K =64Kbits
- Each increase by two in associativity doubles the number of blocks per set and halves the number of sets – decreases the size of index by 1 bit and increase the size of tag by 1 bit
- Two-way set-associative – 2K sets and the total number of tag bits: (28-11)2x2K =68Kbits
- Four-way set-associative – 1K sets and the total number of tag bits: (28-10)4x1K =72Kbits
- Full set-associative – one set with 4K blocks, tag is 28 bits and the total number of tag bits: 28x4K =112Kbits

---

Which block to replace?

- Least Recently Used (LRU): The block replaced is the one that has been unused for the longest time.
- Keep track when each element in a set was used relative to other elements in the set.

Performance



Decreasing miss penalty with multilevel caches

- Add a second level cache:
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache
- Example:
  - CPI of 1.0 on a 5 GHz machine with a 2% miss rate, 100ns DRAM access
  - Adding 2nd level cache with 5ns access time decreases miss rate to 0.5%. How much faster will the processor be?

Performance

- The miss penalty to main memory is:

$$\frac{100\text{ ns}}{0.2\ \dfrac{\text{ns}}{\text{Clock cycle}}} = 500\text{ clock cycles}$$

- The effective CPI with one level of caching is given:

Total CPI = CPI + Memory-stall cycles per instruction

- For one level of caching:

  Total CPI = 1 + 2% x 500 = 11

- For two level of caching a miss in primary cache leads to a secondary cache or main memory. The miss penalty for an access to the second-level cache:

$$\frac{5\text{ ns}}{0.2} = 25\text{ clock cycles}$$

Performance

- Total CPI = 1 Primary stalls per instr. + Secondary stalls per instr. =
  = 1 + 2%x25+0.5%x500 = 4

The processor with the secondary cache is faster by:

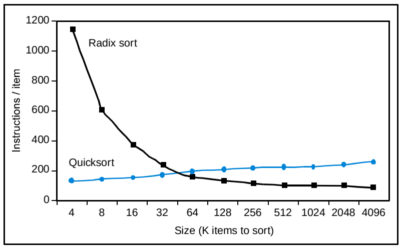
11/4 = 2.8

Improvements

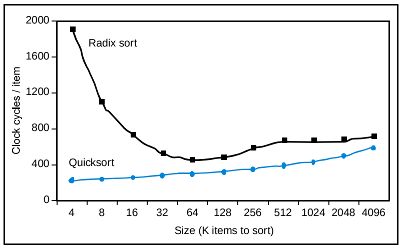
- Using multilevel caches:
  - try and optimize the hit time on the 1st level cache
    - Small and fast
  - try and optimize the miss rate on the 2nd level cache
    - large
- Comparing single to multiple level cache:
  - The primary cache is smaller and uses smaller block size
  - Secondary cache – larger, uses a larger block

Cache Complexities

- Not always easy to understand implications of caches:



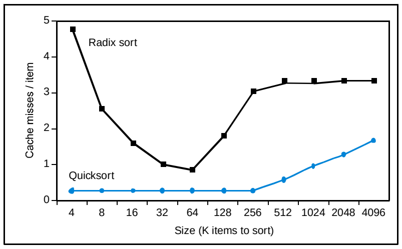
Theoretical behavior of Radix sort vs. Quicksort



Observed behavior of Radix sort vs. Quicksort

Cache Complexities

- Here is why:



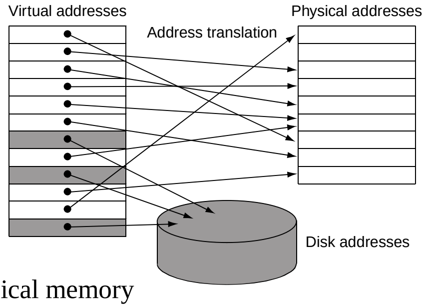
Memory system performance is often critical factor
  - multilevel caches, pipelined processors, make it harder to predict outcomes
  - Compiler optimizations to increase locality sometimes hurt ILP

- Difficult to predict best algorithm: need experimental data

---

Virtual Memory

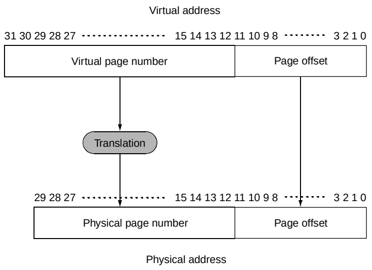
- Main memory can act as a cache for the secondary storage (disk) –
- To allow efficient and safe sharing of memory among multiple programs
- Program to exceed the size of memory



- Advantages:
  - illusion of having more physical memory
  - program relocation
  - Protection – by address translation

Pages: virtual memory blocks

- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty, thus pages should be fairly large
    - Typical 4-16KB, new computers 32-64KB, but embedded systems 1KB
  - reducing page faults is important, LRU is worth the price, fully associative
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use writeback



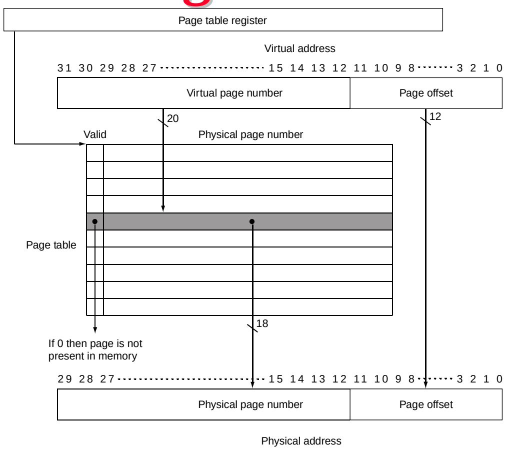
Page Tables

- Pages are located using a Page Table. Each program has its own page table.

- The page table, together with the program counter and registers, specifies the *state* of a program

- The *state* or the *process* is saved when the processor is to be used by another program

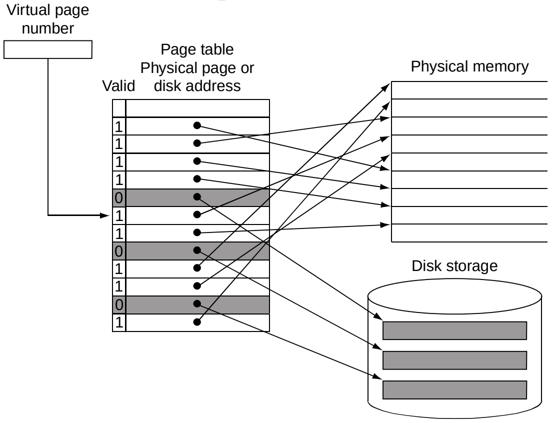
Page Tables



The page table register indicates the location of the page table in memory

---

Page Tables



The page table maps each page in virtual memory to either a page in main memory or a page stored on disk, which is the next level of hierarchy

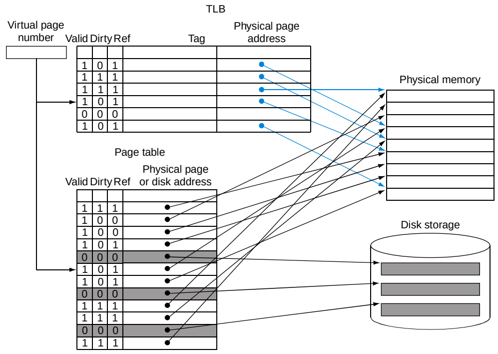
Least Recently Used - LRU

- Implementing a complete accurate LRU is too expensive, since it requires updating a data structure on every memory reference
- Most operating systems approximate LRU by a *use bit* or *reference bit*
- *Reference bit* is set whenever a page is accessed
- The operating system periodically clears the *reference bits* and later records them

---

Making Address Translation Fast

- A cache for address translations: *translation lookaside buffer*



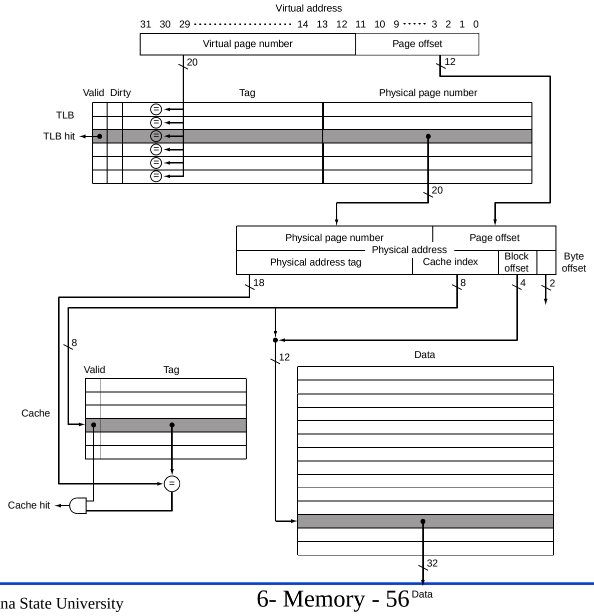
Typical values: 16-512 entries,
miss-rate: .01% - 1%
miss-penalty: 10 – 100 cycles

Intrinsity FastMATH

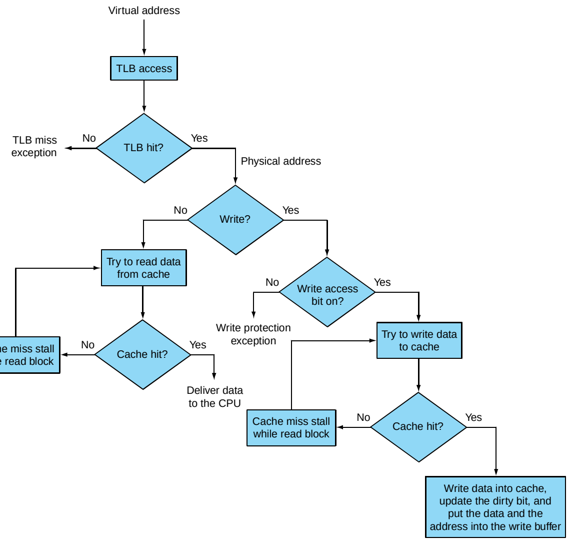
- The memory uses 4KB pages, 32-bit address
- The virtual page number is 20
- The physical address is the same as virtual address
- The TLB contains 16 entries. TLB – full associative
  - Each entry is 64 bits: 20-bit tag, 20-bit physical page number, a valid bit, a dirty bit and other bookkeeping bits

---

TLBs and Caches



TLBs and caches



TLB, virtual memory and cache

TLB	Page Table	Cache	Possible? If so, under what circumstance?
hit	hit	miss	Possible
miss	hit	hit	TLB misses, after retry data is found in cache
miss	hit	miss	TLB misses, after retry data misses in cache
miss	miss	miss	TLB misses, followed by a page fault, after retry data misses in cache
hit	miss	miss	Impossible: cannot have a translation in TLB if page is not in present memory
hit	miss	hit	Impossible: cannot have a translation in TLB if page is not in present memory
miss	miss	hit	Impossible: data cannot be allowed in cache if the page is not in memory

Implementing Protection with Virtual Memory

- Virtual memory allows sharing of a single memory by multiple programs, while providing memory protection among processes and operating system
- Three characteristics:
  - Support at least two modes – user process or operating system processor (supervisor)
  - Provide of processor state that a user processor can read but not write: user/supervisor bit, page table pointer, TLB
  - Provide mechanisms for processor to go from user mode to supervisor mode (by a system call exception) and vice versa.

Protection with Virtual Memory

- Prevent processes from reading the data of another process
- Each process has its own virtual address space
- The operating system keeps the page tables organized so that the independent virtual pages map to disjoint physical pages
- The user process is not able to change the page table mapping. The OP is able to change it.
  - Place the page table in the protected address space of the OP

Protection with Virtual Memory

- Operating System assists processes to share data
- The write access bit can be used to restrict the sharing to just reading (changed only by Operating System)
- To allow process P1 to read a page owned by P2, P2 asks the OS to create a page entry for a virtual page on P1's address space that points to the same physical page that P2 wants to share
- Any bits that determines the access rights for a page must be included in both the page table and TLB. Why?

Context Switch

- When OS changes from running process P1 to P2
- Ensure that P2 cannot get access to page tables of P1
- Without TLB: change the page table register to point to P2's page table
- With TLB: clear TLB entries of P1 and load from P2
  - If the switch rate is high – leads to inefficiency.
    - If P1 is loaded again it will have a high rate of TLB misses
  - Solution: use a *Process Identifier* – it identifies the running process, it is kept in a register loaded by OS when switches processes, it is used concatenated with the tag portion of TLB, therefore no deed to clear the TLB

TLB Misses and Page Faults

- TLB miss can indicate:
  - The page is present in the memory, we need only create the missing TLB entry
  - The page is not present in memory, need to transfer control to the OS to deal with a page fault
- In case of TLB miss, check the matching page table entry – valid bit if on or off (page fault)
- A TLB can be handled in software or hardware
  - In MIPS is handled in software: brings the page table entry from memory than reexecutes the instruction that caused the TLB miss
- In both TLB miss and page fault – use exception to interrupt the active process, transfer control to OS, later resume the execution of the interrupted process
- When there is TLB miss or page fault – the exception must be asserted by the end of the same clock cycle used to access the memory – this stops subsequent memory writes by deasserting the write control line to the memory

Page Faults

- The OS:
  - Look up the page table entry using the virtual address and find the location of the referenced page on disk
  - Chose a physical page to replace; if the chosen one is dirty, it must be written out to disk, before replacing it
  - Start a read to bring the referenced page from disk into the chosen physical page. This takes millions of processor clock cycles. In the mean time the OS selects another process to execute. Why is this possible?
- When the read from disk is completed: OS restores the state of the process, reset the processor from kernel to user mode, restore the program counter. Then the user process reexecutes the instruction the caused the page fault.

Page Faults for data access

- Difficult to be handled compared to instruction page faults
  - Data page faults occur in the middle of instructions
  - The instruction cannot be completed before handling the exception
  - After handling the exception, the instruction must be restarted
- Making instruction *restartable* is easy in an architecture like MIPS. Because each instruction can write only one data item at the end – so we can prevent the writing
- For processors that touch many memory locations making instructions *restartable* is much more difficult
  - In such cases instructions cannot be restarted from beginning, but from midstream – requires saving special state, careful detailed coordination between the exception-handling in OS and harware

---

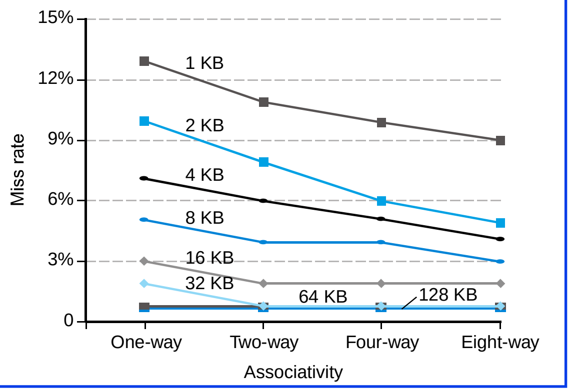
Performance

- If a program routinely accesses more virtual memory than it has physical memory – it would be continuously swapping pages between memory and disk (*thrashing* – its is rare) - it will run very slow
  - Chose a computer with more memory or increase it
  - More complex is to reexamine the algorithm and change the locality to reduce the number of pages needed simultaneously
- More common is TLB misses, TLB 32-64 pages entry, so a processor may access only 64x4KB=0.25 MB
  - Support variable page size. In MIPS : 16KB, 64, 256, 1 MB, 4, 16 and 256 MB pages

Where can a Block be placed?

- The advantage of increasing the degree of associativity is that it usually decreases the miss rate
  - The improvement comes from reducing the competition for the same location
  - The potential disadvantages – increase cost and access time

SPEC2000 benchmark



How is a Block Found?

Associativity	Location Method	Comparison required
Direct Mapped	index	1
Set associative	Index the set, search among elements	Degree of associativity
Full	Search all cache entries	Size of the cache
	Separate lookup tables	0

- Including L2 cache on chip enables much higher associativity. Full associativity is only for small sizes
- In virtual memory the choice of full associativity:
  - Misses are very expensive
  - Allow software to use sophisticated replacement schemes that reduce miss rate
  - The full mapp can be easily indexed with no extra hardware and no searching
  - The large page size means the page table size overhead is small
- In TLB – set associative. Some systems use direct-mapped to reduce access time. This depends on whether the cache is on-chip, the technology used, the critical role of access time

Which Block should be replaced on a cache miss?

- In full associativity – all blocks are candidate for replacement
  - Least Recently Used – is costly to implement for hierarchies with more than a small degree of associativity (2-4). It often approximated
  - Random – in higher levels of associativity
  - As the caches become larger the miss rate for both strategies falls
- In virtual memory some form of LRU is always approximated since even a small reduction in miss rate is important

---

What happens on a write?

- Write-through: the information is written to both the block in the cache and to the block in the lower level of memory hierarchy. Caches use this scheme.
  - Misses are simpler and cheaper because they never require a block to be written back to lower level
  - Easy to be implemented than write-back
- Write-back (copy-back): the information is written only to the block in the cache. The modified block is written to the lower level of hierarchy only when replaced. Virtual memory always uses write-back
  - Individual words can be written at the rate of cache
  - Multiple writes within the a block require only one write to the lower level
  - When blocks are written, the system can make effective use of a high bandwidth transfer
- In virtual memory – only write-back is practical

Memory Hierarchies

- Classification of misses:
  - Compulsory misses: caused by the first access to a block that has never been in the cache – cold start misses
  - Capacity misses: when the cache cannot contain all the blocks needed during execution of a program. When blocks are replaced and then later retrieved
  - Conflict misses: in set-associative or direct-mapped when multiple blocks compete for the same set. Collision misses

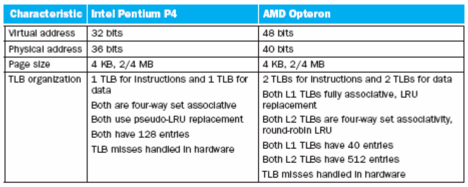
Design change	Effect on miss rate	Possible negative effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease miss rate due to conflict	May increase access time
Increase block size	Decrease miss rate for a wide range of block sizes due to spatial locality	Increases miss penalty. Very large block could increase miss rate

---

Memory Hierarchies

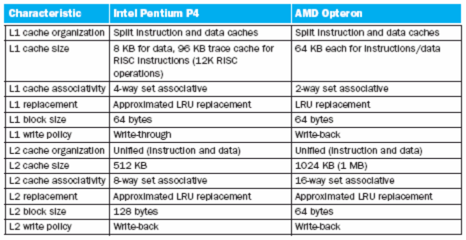
- The challenge is that every change that potentially improves the miss rate can also affect the overall performance

Modern Systems



Characteristic	Intel Pentium P4	AMD Opteron
Virtual address	32 bits	48 bits
Physical address	36 bits	40 bits
Page size	4 KB, 2/4 MB	4 KB, 2/4 MB
TLB organization	1 TLB for instructions and 1 TLB for data Both are four-way set associative Both use pseudo-LRU replacement Both have 128 entries TLB misses handled in hardware	2 TLBs for instructions and 2 TLBs for data Both L1 TLBs fully associative, LRU replacement Both L2 TLBs are four-way set associativity, round-robin LRU Both L1 TLBs have 40 entries Both L2 TLBs have 512 entries TLB misses handled in hardware

FIGURE 7.34 Address translation and TLB hardware for the Intel Pentium P4 and AMD Opteron. The word size sets the maximum size of the virtual address, but a processor need not use all bits. The physical address size is independent of word size. The P4 has one TLB for instructions and a separate identical TLB for data, while the Opteron has both an L1 TLB and an L2 TLB for instructions and identical L1 and L2 TLBs for data. Both processors provide support for large pages, which are used for things like the operating system or mapping a frame buffer. The large-page scheme avoids using a large number of entries to map a single object that is always present.

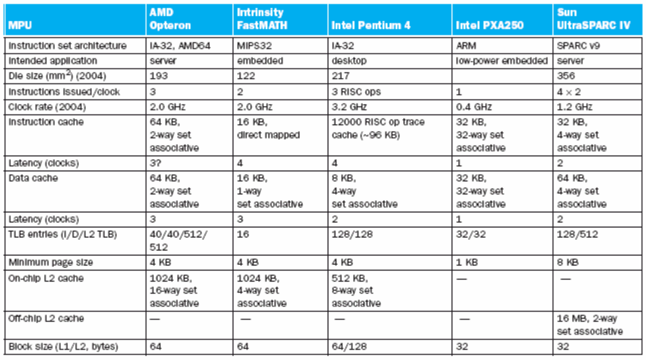


Characteristic	Intel Pentium P4	AMD Opteron
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	8 KB for data, 96 KB trace cache for RISC instructions (12K RISC operations)	64 KB each for instructions/data
L1 cache associativity	4-way set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-through	Write-back
L2 cache organization	Unified (instruction and data)	Unified (instruction and data)
L2 cache size	512 KB	1024 KB (1 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	128 bytes	64 bytes
L2 write policy	Write-back	Write-back

FIGURE 7.35 First-level and second-level caches in the Intel Pentium P4 and AMD Opteron. The primary caches in the P4 are physically indexed and tagged; for a discussion of the alternatives, see the Elaboration on page 527.

Modern Systems

- Things are getting complicated!

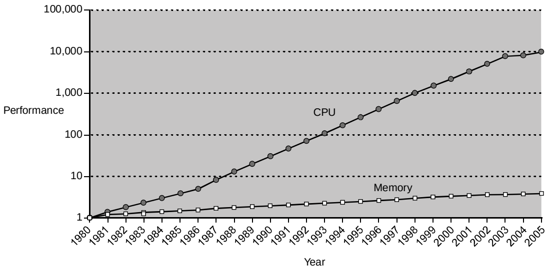


MPU	AMD Opteron	Intrinsity FastMATH	Intel Pentium 4	Intel PXA250	Sun UltraSPARC IV
Instruction set architecture	IA-32, AMD64	MIPS32	IA-32	ARM	SPARC v9
Intended application	server	embedded	desktop	low-power embedded	server
Die size (mm^2) (2004)	193	122	217		356
Instructions issued/clock	3	2	3 RISC ops	1	4 × 2
Clock rate (2004)	2.0 GHz	2.0 GHz	3.2 GHz	0.4 GHz	1.2 GHz
Instruction cache	64 KB, 2-way set associative	16 KB, direct mapped	12000 RISC op trace cache (~96 KB)	32 KB, 32-way set associative	32 KB, 4-way set associative
Latency (clocks)	3?	4	4	1	2
Data cache	64 KB, 2-way set associative	16 KB, 1-way set associative	8 KB, 4-way set associative	32 KB, 32-way set associative	64 KB, 4-way set associative
Latency (clocks)	3	3	2	1	2
TLB entries (I/D/L2 TLB)	40/40/512/512	16	128/128	32/32	128/512
Minimum page size	4 KB	4 KB	4 KB	1 KB	8 KB
On-chip L2 cache	1024 KB, 16-way set associative	1024 KB, 4-way set associative	512 KB, 8-way set associative	—	—
Off-chip L2 cache	—	—	—	—	16 MB, 2-way set associative
Block size (L1/L2, bytes)	64	64	64/128	32	32

FIGURE 7.36 Desktop, embedded, and server microprocessors in 2004. From a memory hierarchy perspective, the primary differences between categories is the L2 cache. There is no L2 cache for the low-power embedded, a large on-chip L2 for the embedded and desktop, and 16 MB off chip for the server. The processor clock rates also vary: 0.4 GHz for low-power embedded, 1 GHz or higher for the rest. Note that UltraSPARC IV has two processors on the chip.

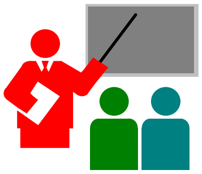
Some Issues

- Processor speeds continue to increase very fast
  — much faster than either DRAM or disk access times



- Design challenge: dealing with this growing disparity
  - Prefetching? 3rd level caches and more? Memory design?

Summary



- Memory Hierarchy
- Caches – Hi vs. Miss
- Associativity
- Virtual Memory
- Translation Lookaside Buffer
- The challenge is that every change that potentially improves the miss rate can also affect the overall performance