Computer Architecture and Organization

Chapter 7
Memory Hierarchy

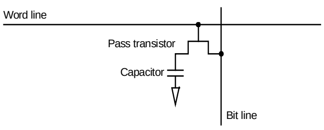
Henk Corporaal / Ben Juurlink
Delft University of Technology
Spring 2000
Email: h.corporaal@et.tudelft.nl
URL : http://cardit.et.tudelft.nl/~heco/courses/et2-010

Topics

- Memories: review
- Memory Hierarchy: why?
- Basics of caches
- Measuring cache performance
- Improving cache performance
- Framework for memory hierarchies
- Virtual memory
- Pentium Pro and PowerPC 604 memory hierarchies

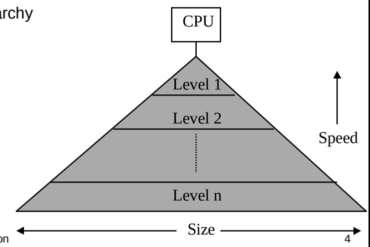
Memories: Review

- 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)



Memory Hierarchy, why?

- Users want large and fast memories!
  SRAM access times are 2 - 25ns at cost of $100 to $250 per Mbyte.
  DRAM access times are 60-120ns at cost of $5 to $10 per Mbyte.
  Disk access times are 10 to 20 million ns at cost of $.10 to $.20 per Mbyte. **1997**
- Try and give it to them anyway
  - build a memory hierarchy



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

Exploiting locality

Cache

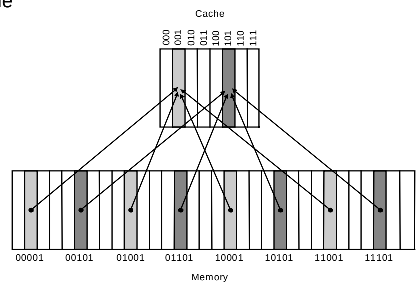
- Two issues:
  - How do we know if a data item is in the cache?
  - If it is, how do we find it?
- Our first example:
  - *block size* is one word of data
  - "*direct mapped*"

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

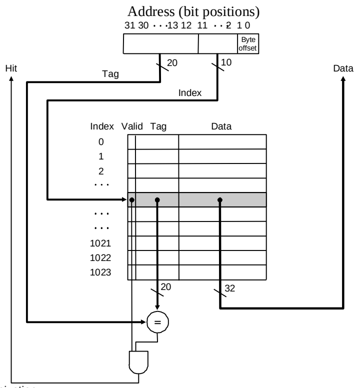
- Mapping: address is modulo the number of blocks in the cache



Direct Mapped Cache

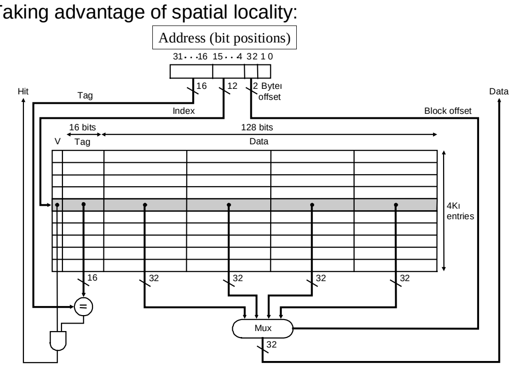
- For MIPS:

What kind of locality are we taking advantage of?



Direct Mapped Cache

- Taking advantage of spatial locality:

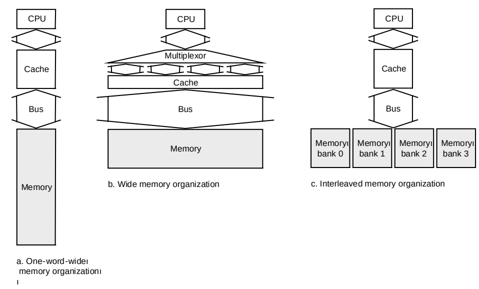


Hits vs. Misses

- Read hits
  - this is what we want!
- Read misses
  - stall the CPU, fetch block from memory, deliver to cache, restart the load instruction
- Write hits:
  - can replace data in cache and memory (*write-through*)
  - write the data only into the cache (*write-back* the cache later)
- Write misses:
  - read the entire block into the cache, then write the word (*allocate* on write miss)
  - do not read the cache line; just write to memory (*no allocate* on write miss)

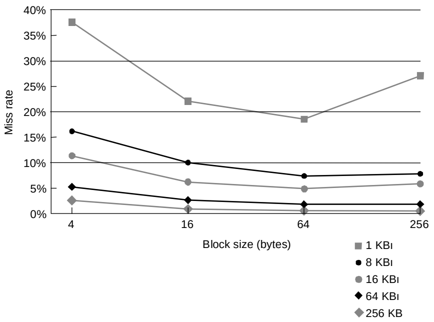
Hardware Issues

- Make reading multiple words easier by using banks of memory



Performance

- Increasing the block size tends to decrease miss rate:



computer architecture and organization 13

Performance

- 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%

computer architecture and organization 14

Performance

- Simplified model:

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

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

computer architecture and organization 15

Performance

$T_{exec} = N_{inst} \bullet CPI \bullet T_{cycle}$
with
$CPI = CPI_{ideal} + CPI_{stall}$
CPI_{stall} = %reads • $missrate_{read}$ • $misspenalty_{read}$ + %writes • $missrate_{write}$ • $misspenalty_{write}$

or:

$T_{exec} = (N_{normal\text{-}cycles} + N_{stall\text{-}cycles}) \bullet T_{cycle}$
with
$N_{stall\text{-}cycles} = N_{reads} \bullet missrate_{read} \bullet misspenalty_{read} + N_{writes} \bullet missrate_{write} \bullet misspenalty_{write}$
(+ Write-buffer stalls)

computer architecture and organization 16

Performance example

- Assume GCC application (pg 311)
  - Icache missrate 2%
  - Dcache missrate 4%
  - CPI ideal is 2.0
  - Misspenalty 40 cycles
- Calculate CPI

CPI = 2.0 + CPIstall
CPIstall = Instruction-miss cycles + Data-miss cycles
Instruction-miss cycles = Ninstr x 0.02 x 40 = 0.80 Ninstr
Data-miss cycles = Ninstr x %ld-st x 0.04 x 40
%ld-st = 36 %
Slowdown: 1.68 !!

computer architecture and organization 17

Performance example (cont'd)

What if ideal processor had CPI = 1.0 (instead of 2.0)

- Slowdown would be 2.36 !

What if processor is clocked twice as fast

- => penalty becomes 80 cycles

- CPI = 4.75
- Speedup = N.CPIa.Tclock / (N.CPIb.Tclock/2) = 3.36 / (4.75/2)
- Speedup is not 2, but only 1.41 !!

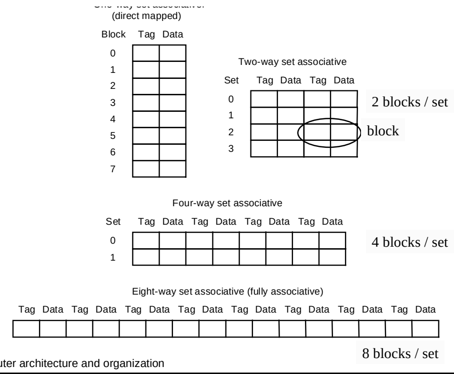
computer architecture and organization 18

Improving performance

- Two ways of improving performance:
  - decreasing the miss ratio: *associativity*
  - decreasing the miss penalty: *multilevel caches*

What happens if we increase block size?

Decreasing miss ratio with associativity

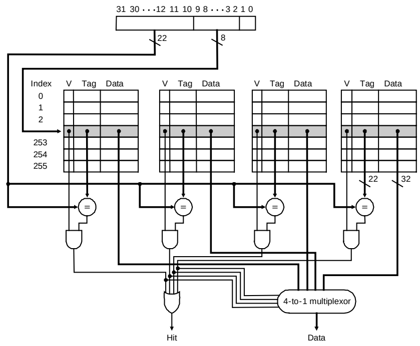


Decreasing miss ratio with associativity

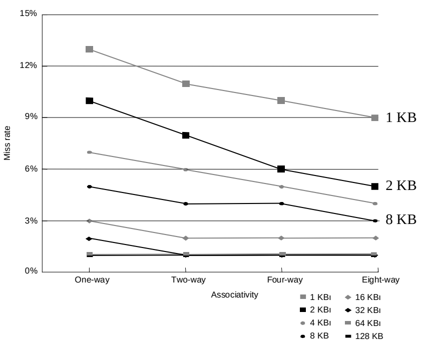
- *Compared to direct mapped, give a series of references that:*
  - *results in a lower miss ratio using a 2-way set associative cache*
  - *results in a higher miss ratio using a 2-way set associative cache*

assuming we use the "least recently used" (LRU) replacement strategy

An implementation: 4 way associative

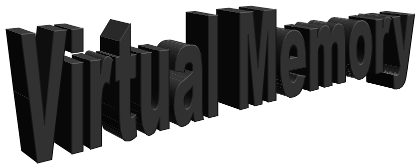


Performance



Decreasing miss penalty with multilevel caches

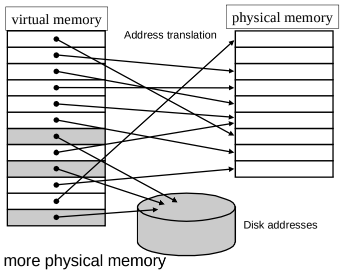
- Add a second level cache:
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache
- Example:
  - CPI of 1.0 on a 500Mhz machine with a 5% miss rate, 200ns DRAM access
  - Adding 2nd level cache with 20ns access time decreases miss rate to 2%
- Using multilevel caches:
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache



The illusion of one big (protected) memory

Virtual Memory

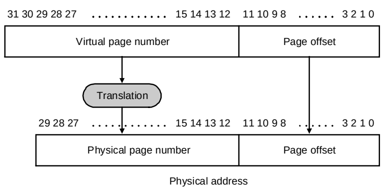
- Main memory can act as a cache for the secondary storage (disk)



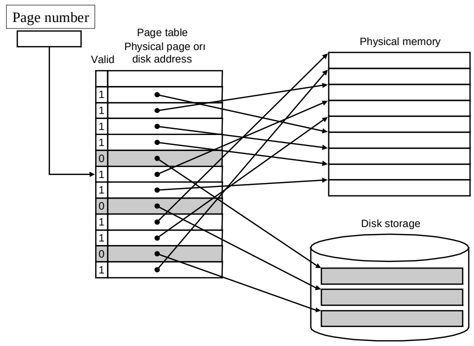
- Advantages:
  - illusion of having more physical memory
  - program relocation
  - protection

Pages: virtual memory blocks

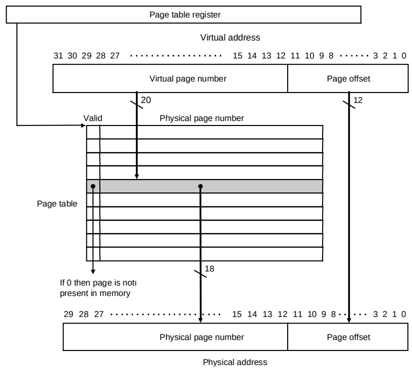
- Page faults: the data is not in memory, retrieve it from disk
  - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
  - reducing page faults is important (LRU is worth the price)
  - can handle the faults in software instead of hardware
  - using write-through is too expensive so we use writeback



Page Tables

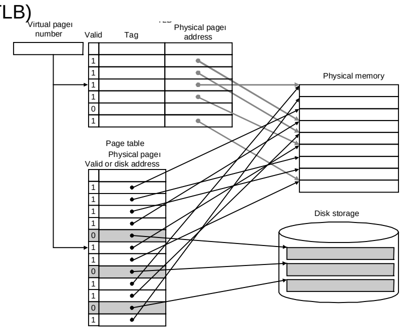


Page Tables

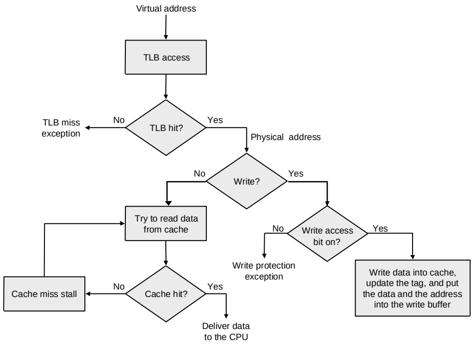


Making Address Translation Fast

- A cache for address translations: translation lookaside buffer (TLB)



TLBs and caches



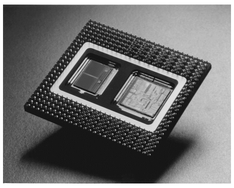
Modern Systems

- Very complicated memory systems
- Virtual memory:

Characteristic	Intel Pentium Pro	PowerPC 604
Virtual address	32 bits	52 bits
Physical address	32 bits	32 bits
Page size	4 KB, 4 MB	4 KB, selectable, and 256 MB
TLB organization	A TLB for instructions and a TLB for data Both four-way set associative Pseudo-LRU replacement Instruction TLB: 32 entries Data TLB: 64 entries TLB misses handled in hardware	A TLB for instructions and a TLB for data Both two-way set associative LRU replacement Instruction TLB: 128 entries Data TLB: 128 entries TLB misses handled in hardware

Modern Systems

- First level Cache organization



Pentium Pro dual chip module

Characteristic	Intel Pentium Pro	PowerPC 604
Cache organization	Split instruction and data caches	Split intruction and data caches
Cache size	8 KB each for instructions/data	16 KB each for instructions/data
Cache associativity	Four-way set associative	Four-way set associative
Replacement	Approximated LRU replacement	LRU replacement
Block size	32 bytes	32 bytes
Write policy	Write-back	Write-back or write-through

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
- Trends:
  - synchronous SRAMs (provide a burst of data)
  - redesign DRAM chips to provide higher bandwidth or processing
  - restructure code to increase locality
  - use prefetching (make cache visible to ISA)

Exercises

From Chapter seven:

- 7.2, 7.3
- 7.7 - 7.10
- 7.20, 7.21
- 7.27, 7.32