Cache Basics

CMPSCI 635
Modern Computer Architecture
Univ. of Mass / Amherst

Lecture Outline

- A brief history of computer memories
- The memory hierarchy
- Cache Basics
  - Motivation
  - Cache metrics
  - Cache implementation
  - Set associativity of cache

Early Memories

- Vacuum tubes, relays
- Cathode ray tubes
- Acoustical delay lines
- Magnetic drums
- Magnetic cores
- Punch cards
- Open-reel tape

Modern Memory

- Semiconductor based RAM
- Magnetic disk
- Magnetic or optical tape
- Optical disk (CDs, DVDs)
- Magneto-optical disk
- Flash memories

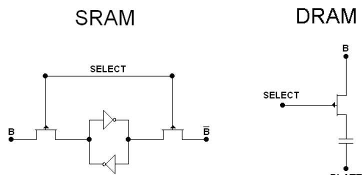
Memory Technology

- Memory technology is mature
- Any new technology has a lot of catching up to do to be competitive
- New memory technologies have become much more difficult to introduce
- Memory needs are currently driven by entertainment industry

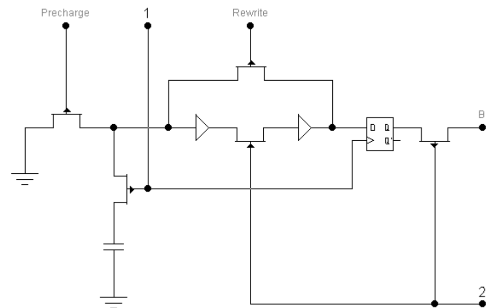
DRAM vs. SRAM

- Dynamic RAM (DRAM) is implemented using capacitors to store charge
- DRAM must be periodically refreshed due to capacitor leakage
- SRAM uses flip-flops that do not need refreshing, thus SRAM is faster
- SRAM is more complex and expensive

DRAM vs. SRAM

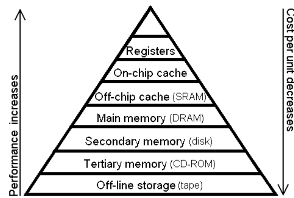


DRAM in Detail

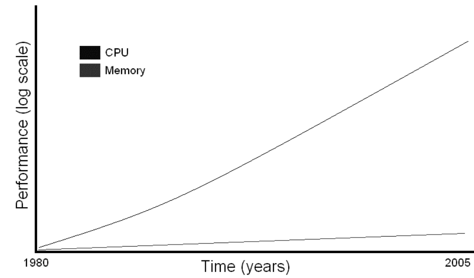


Memory Hierarchy

We use a memory hierarchy so that we can get varying amounts of cost per performance in a system:



"The Performance Gap" (myth)



"The Performance Gap" (truth)

- CPU performance cannot increase exponentially with respect to memory
- Access time to main memory is typically ~100 cycles
- This gap is very significant
- Opening gaps are between on and off-chip memory, and registers and L1 cache

Caches

- Probabilistic attempt to keep useful data "closer" to the CPU
- Important because of the performance distance between CPU and main memory
- Higher in the memory hierarchy than main memory, so it is faster and more costly
- Operates on the assumption that memory accesses exhibit *locality*

Basic Properties of Cache

- Inclusion: copies of a value exist in all levels below
- Coherence: all copies are the same
- Locality: programs access a limited portion of the address space during any window in time
- May not strictly hold at all times

Cache Parameters

- Access time (CPU to memory)
- Size (bytes)
- Cost per unit (byte)
- Transfer bandwidth (sustained)
- Unit of transfer a.k.a. "line size" (bytes moved)
- Set associativity

Hit ratio

- Probability that an access hits at a given level
- A hit is an access to something already in cache
- Miss ratio = (1 – Hit ratio)
- Access frequency is the product of the hit ratio for a given level and the miss ratios of all levels above it
- Cache performance is not completely dependent on hit ratio

Average Access Time

- Sum of access frequencies for all levels multiplied by their corresponding access times
  - Assumes no bypass
  - Assumes that data resides in the lowest level (data is always present)
- Good design depends on simulation

Average Access Time

Average Memory Access Time =

L1 Hit time + L1 Miss rate * L1 Miss penalty

L1 Miss penalty =

L2 Hit time + L2 Miss rate * L2 Miss penalty

L2 Miss penalty =

L3 Hit time + L3 Miss rate * L3 Miss penalty

Example

In 1000 memory references there are 40 misses in L1 and 20 misses in L2. Assume all are reads with no bypass.

L1: Hit time = 1 cycle

L2: Hit time = 10 cycles

Miss penalty = 100 cycles

What is the average memory access time?

Answer

L1 Local miss rate = L1 Local misses / Total references = 40/1000 = 4%

L2 Local miss rate = L2 Local misses / L1 Local misses = 20/40 = 50%

Average Memory Access Time =

L1 Hit Time + L1 Miss Rate *

(L2 Hit Time + L2 Miss Rate * L2 Miss Penalty) =

1 + 4% * (10 + 50% * 100) =

3.4 clock cycles

Reducing Avg. Access Time

- Make memory faster (costly)
- Reduce cache miss rate
- Try to exploit more *locality*
  - Change cache parameters
  - Optimize code
- Make cache bigger
- More elaborate schemes (later)

Programmed Transfer

- We explicitly specify transfers from main to secondary memory
- We can also explicitly specify transfers from registers to memory
- No language exists to provide information for better management of cache
- Compilers can be used to optimize for the cache

Linguistic Issues

- Languages restrict compiler context
- Higher level constructs could enable scheduling of more transfers
- Compiler might reorder accesses to improve locality
- Currently done for loops; between functions is hard, irregular structures are even harder

Types of Locality

- Temporal: recently accessed data tends to be accessed again in the near future
- Spatial: data accesses tend to be close together
- Sequential: instructions (also some data) tend to be accessed in consecutive order

Spatial Locality

- Widely found in codes
  - Array accesses
  - Instruction streams
- Spatial locality implies temporal locality, but not vice versa
- Exploited by larger line size
- Similar to sequential locality

Temporal Locality

- Tends to hold for many codes
  - Arrays in loops
  - Linked lists
  - Local variables, constants
- Exploited by smaller line size
- Does NOT imply spatial locality

Sequential Locality

- Especially true of instructions
- Subset of spatial locality
- Locality differences between data and instructions argue for split (instruction separate from data) caches

Distinguishing Localities

- Spatial hit: A hit that occurs on a word already in cache that has not been previously accessed
- Temporal hit: A hit that occurs on a word already in cache that has been previously accessed
- Eviction erases history with respect to these definitions

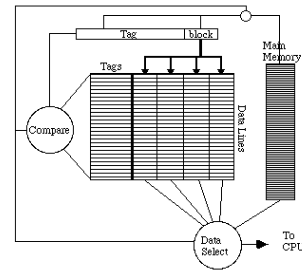
Lines in Cache

- A "line" stores a memory "block," which is comprised of contiguous words
- Increased line sizes increase the probability that contiguous data is moved into the cache on a miss
- Increased number of lines decrease the probability that a specific line is evicted

Basic Cache Structure

- Comprised of lines, which store contiguous blocks
- Each line has a tag (high order bits) to keep track of which memory block is present
- Position in cache is determined by low order bits (except in fully associative)

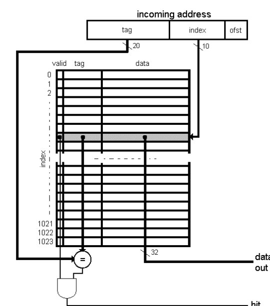
Fully Associative Cache



Fully Associative (K=L)

- Pros
  - Any memory block can go anywhere in cache
  - Cache can be fully utilized
  - High hit rate, does not suffer conflict misses
- Cons
  - Requires one comparator per line (impractical)
  - Fan-in delays for compare and access
  - Replacement algorithms become impractical

Direct Mapped



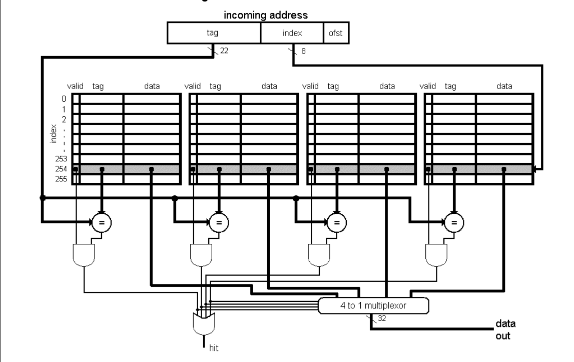
Direct Mapped (S=L)

- Pros
  - Only one comparator
  - Direct fetch and test of tag
  - Fast access
- Cons
  - Low utilization
  - Many conflict misses
  - Lower hit ratio

Cache Associativity

- Cache can be divided into sets
- K-ways divide cache into sets with K lines per set
- Number of sets is S = L/K where L is number of lines in the cache
- A block may go anywhere its associated set
- Associativity is directly related to conflict misses

4-way Set Associative



K-way Associative

- An effective balance between the speed of direct mapped and the performance of fully assoc.
- K-wide comparator almost as fast and cheap as direct mapped
- Hit ratio is almost like fully associative
- Up to K blocks with the same mapping may be in cache at once without conflicting

Example

- 4-way set associative cache with 16K 4-word lines (1 word = 4 bytes)
- 256K byte capacity
- 4K sets
- 4 ways (4 comparators)
- Each memory address can map to 4 locations in cache

Next Lecture

- More on reasons for miss
- Cache traces
  - Running (an example on)
  - Generating
  - Analyzing
  - Statistical correctness of