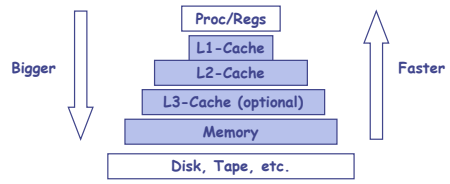
Lecture 13: Cache Basics and Cache Performance

Memory hierarchy concept, cache design fundamentals, set-associative cache, cache performance, Alpha 21264 cache design

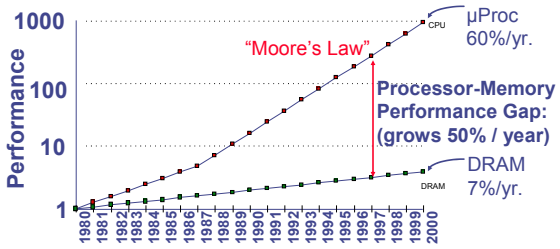
What Is Memory Hierarchy

- A typical memory hierarchy today:



- Here we focus on L1/L2/L3 caches and main memory

Why Memory Hierarchy?



- 1980: no cache in µproc; 1995 2-level cache on chip (1989 first Intel µproc with a cache on chip)

Generations of Microprocessors

- Time of a full cache miss in instructions executed:

1st	Alpha:	340 ns/5.0 ns = 68 clks x 2 or 136
2nd	Alpha:	266 ns/3.3 ns = 80 clks x 4 or 320
3rd	Alpha:	180 ns/1.7 ns =108 clks x 6 or 648

- 1/2X latency x 3X clock rate x 3X Instr/clock ⇒ 4.5X

Area Costs of Caches

Processor	% Area *(-cost)*	%Transistors *(-power)*
Intel 80386	0%	0%
Alpha 21164	37%	77%
StrongArm SA110	61%	94%
Pentium Pro	64%	88%
▪ 2 dies per package: Proc/I$/D$ + L2$		
Itanium		92%

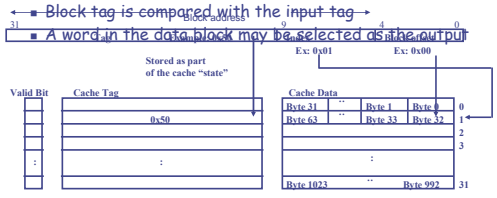
- Caches store redundant data only to close performance gap

What Is Exactly Cache?

- Small, fast storage used to improve average access time to slow memory; usually made by SRAM
- Exploits locality: spatial and temporal
- In computer architecture, almost everything is a cache!
  - Register file is the fastest place to cache variables
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information?
  - Branch-target buffer can be implemented as cache
- Beyond architecture: file cache, browser cache, proxy cache
- Here we focus on L1 and L2 caches (L3 optional) as buffers to main memory

Example: 1 KB Direct Mapped Cache

- Assume a cache of 2^N bytes, 2^K blocks, block size of 2^M bytes; N = M+K (#block times block size)
  - (32 - N)-bit cache tag, K-bit cache index, and M-bit cache
- The cache stores tag, data, and valid bit for each block
  - Cache index is used to select a block in SRAM (Recall BHT, BTB)
  - Block tag is compared with the input tag
  - A word in the data block may be selected as the output



For Questions About Cache Design

Block placement: Where can a block be placed?

Block identification: How to find a block in the cache?

Block replacement: If a new block is to be fetched, which of existing blocks to replace? (if there are multiple choice)

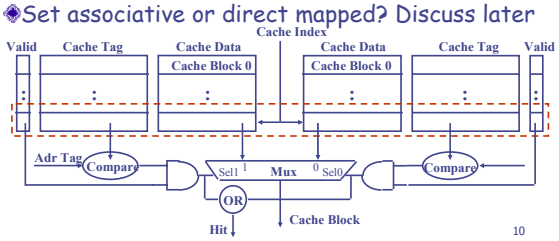
Write policy: What happens on a write?

Where Can A Block Be Placed

- What is a block: divide memory space into blocks as cache is divided
  - A memory block is the basic unit to be cached
- Direct mapped cache: there is only one place in the cache to buffer a given memory block
- N-way set associative cache: N places for a given memory block
  - Like N direct mapped caches operating in parallel
  - Reducing miss rates with increased complexity, cache access time, and power consumption
- Fully associative cache: a memory block can be put anywhere in the cache

Set Associative Cache

- Example: Two-way set associative cache
  - Cache index selects a set of two blocks
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag comparison
- Set associative or direct mapped? Discuss later



How to Find a Cached Block

- Direct mapped cache: the stored tag for the cache block matches the input tag
- Fully associative cache: any of the stored N tags matches the input tag
- Set associative cache: any of the stored K tags for the cache set matches the input tag

Cache hit latency is decided by both tag comparison and data access

Which Block to Replace?

- Direct mapped cache: Not an issue
- For set associative or fully associative* cache:
  - Random: Select candidate blocks randomly from the cache set
  - LRU (Least Recently Used): Replace the block that has been unused for the longest time
  - FIFO (First In, First Out): Replace the *oldest* block
- Usually LRU performs the best, but hard (and expensive) to implement

*Think fully associative cache as a set associative one with a single set

What Happens on Writes

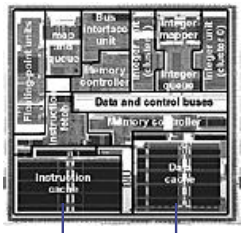
Where to write the data if the block is found in cache?

- Write through: new data is written to both the cache block and the lower-level memory
  - Help to maintain cache consistency
- Write back: new data is written only to the cache block
  - Lower-level memory is updated when the block is replaced
  - A dirty bit is used to indicate the necessity
  - Help to reduce memory traffic

What happens if the block is not found in cache?

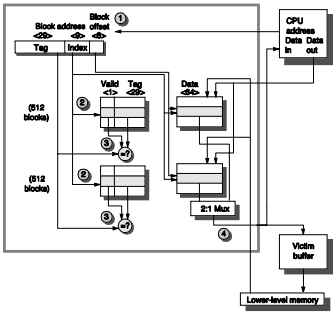
- Write allocate: Fetch the block into cache, then write the data (usually combined with write back)
- No-write allocate: Do not fetch the block into cache (usually combined with write through)

Real Example: Alpha 21264 Caches



- 64KB 2-way associative instruction cache
- 64KB 2-way associative data cache

Alpha 21264 Data Cache



D-cache: 64K 2-way associative

- Use 48-bit virtual address to index cache, use tag from physical address
- 48-bit Virtual=>44-bit address
- 512 block (9-bit blk index)
- Cache block size 64 bytes (6-bit offset)t
- Tag has 44-(9+6)=29 bits
- Writeback and write allocated

(We will study virtual-physical address translation)

Cache performance

- Calculate average memory access time (AMAT)

$$\text{AMAT} = \text{hit time} + \text{Miss rate} \times \text{Miss penalty}$$

  - Example: hit time = 1 cycle, miss time = 100 cycle, miss rate = 4%, than AMAT = 1+100*4% = 5
- Calculate cache impact on processor performance

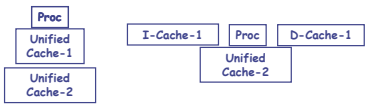
$$\text{CPU time} = (\text{CPU execution cycles} + \text{Memory stall cycles}) \times \text{Cycle time}$$

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory Stall Cycles}}{\text{Instruction}} \right) \times \text{CycleTime}$$

  - Note cycles spent on cache hit is usually counted into execution cycles
- If clock cycle is identical, better AMAT means better performance

Example: Evaluating Split Inst/Data Cache

- Unified vs Split Inst/data cache (Harvard Architecture)



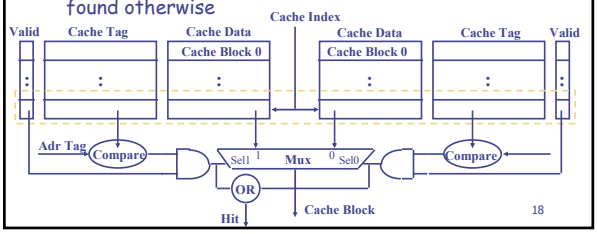
- Example on page 406/407
  - Assume 36% data ops ⇒ 74% accesses from instructions (1.0/1.36)
  - 16KB I&D: Inst miss rate=0.4%, data miss rate=11.4%, overall 3.24%
  - 32KB unified: Aggregate miss rate=3.18%
- Which design is better?
  - hit time=1, miss time=100
  - Note that *data* hit has 1 stall for unified cache (only one port)

$AMAT_{Harvard}$=74%x(1+0.4%x100)+26%x(1+11.4%x100) = 4.24

$AMAT_{Unified}$=74%x(1+3.18%x100)+26%x(1+1+3.18%x100)= 4.44

Disadvantage of Set Associative Cach

- Compare n-way set associative with direct mapped cache:
  - Has n comparators vs. 1 comparator
  - Has Extra MUX delay for the data
  - Data comes after hit/miss decision and set selection
- In a direct mapped cache, cache block is available before hit/miss decision
  - Use the data assuming the access is a hit, recover if found otherwise



Example: Evaluating Set Associative Cache

- Suppose a processor with
  - 16Hz speed, Ideal (no misses) CPI = 2.0
  - 1.5 memory references per instruction
- Two cache organization alternatives
  - Direct mapped, 1.4% miss rate, hit time 1 cycle, miss penalty 75ns
  - 2-way set associative, 1.0% miss rate, increase cycle time by 1.25x, hit time 1 cycle, miss penalty 75ns
- Performance evaluation by AMAT
  - Direct mapped: 1.0 + (0.014 x 75) = 2.05ns
  - 2-way set associative: 1.0 x 1.25 + (0.10 x 75) = 2.00ns
- Performance evaluation by CPU time
  - CPU Time 1 = IC x (2x1.0 + (1.5x0.014x75) = 3.58 IC
  - CPU Time 2 = IC x (2x1.0x1.25 + 1.5x0.010x75)=3.63IC
- Better AMAT does not indicate better CPI time, since non-memory instructions are penalized

Evaluating Cache Performance for Out-of-order Processors

Recall AMAT = hit time + miss rate x miss penalty

- Very difficult to define miss penalty to fit in this simple model, in the context of OOO processors
  - Consider overlapping between computation and memory accesses
  - Consider overlapping among memory accesses for more than one misses
- We may assume a certain percentage of overlapping
  - In practice, the degree of overlapping varies significantly between
  - There are techniques to increase the overlapping, making the cache performance even unpredictable
- Cache hit time can also be overlapped
  - The increase of CPI is usually not counted in memory stall time

Simple Example

Consider an OOO processors into the previous example (slide 18)

- Slow clock (1.25x base cycle time)
- Direct mapped cache
- Overlapping degree of 30%

Average miss penalty = 70% * 75ns = 52.5ns

AMAT = 1.0x1.25 + (0.014x52.5) = 1.99ns

CPU time = ICx(2x1.0x1.25+(1.5x0.014x52.5))=3.60xIC

Compare: 3.58 for in-order + direct mapped, 3.63 for in-order + two-way associative

This is only a simplified example; ideal CPI could be improved by OOO execution