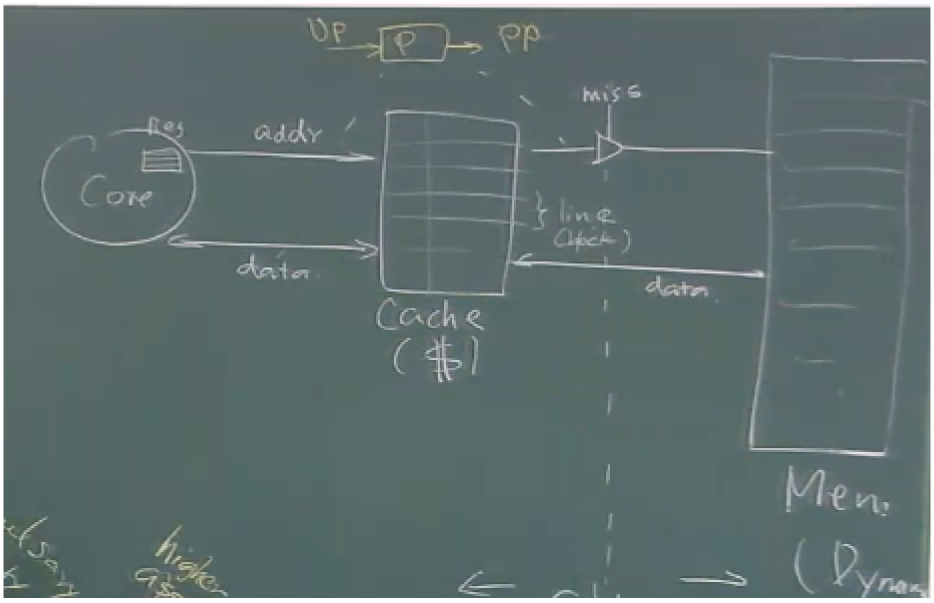
W14D2 Computer Architecture Notes

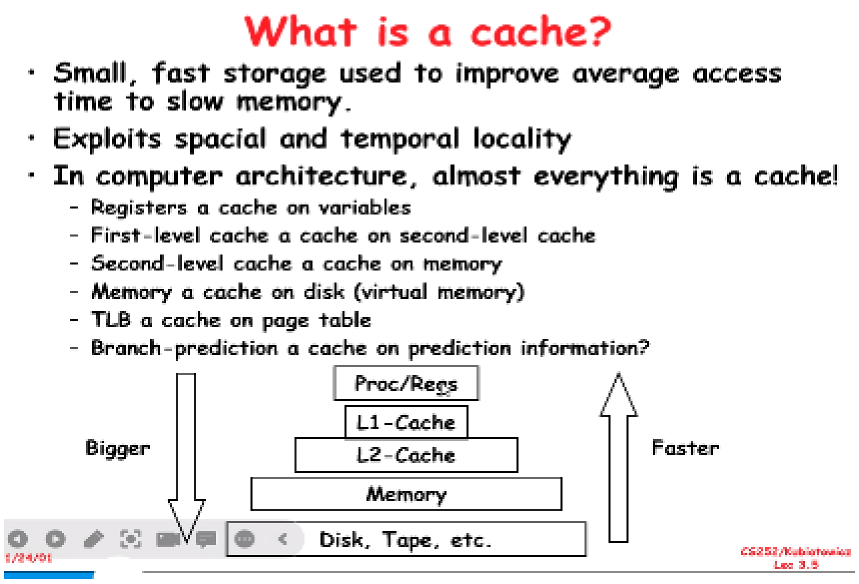
1. Cache/Reg/Mem hierarchy

1.1 Overall description

In computer systems, registers are closest to core, and cache serves as a closer place to keep the warm(namely frequently visited) information/data of memory.

This is an intuitive figure about this hierarchy by Alei:





What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- Exploits spacial 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)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information?

1.2 AMAT

The well-known AMAT rule is:

$$AMAT = T_{hit} + \eta \times T_{pen}$$

Here η stands for the miss rate.

The AMAT rule gives us three main ways to improve the performance of our system:

1. Reducing hit time
2. Reducing miss rate
3. Reducing time penalty (when a cache-miss happens)

1.3 Ways to reduce miss rate

Bear the "4Cs" in mind. Coherence, Compulsory, Capacity, Conflict.

I'm sure that you're familiar with these concepts, for further understanding or revision, you can refer to CAAQA or this note for Arch revision.

It's important that when implementing higher associativity, the capacity miss rate reduces at the cost of increasing conflict misses.

Between Memory and Cache:

- stream buffer(critical first)

Software optimization

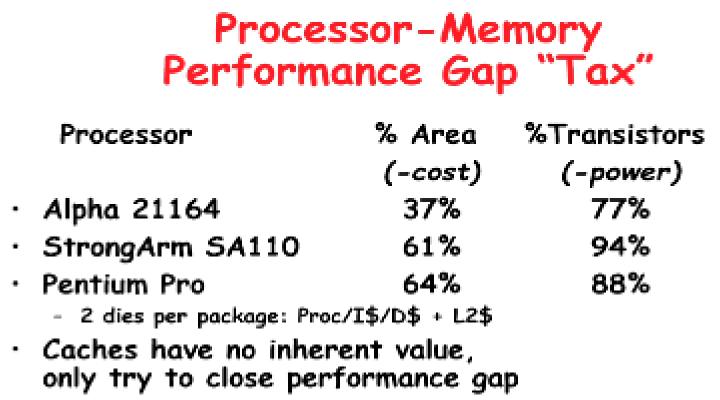
- pre-fetch(on software level)
- switching the orders in loops

1.4 Basic features of memory

The most fundamental and important features are **space locality and time locality** (i.e. the data we fetch from the memory would be more likely to be visited in the next time period, and the data we read/write is within a relatively smaller range, like instructions), this is the rudimentary reason why cache is needed or efficient.

1.5 Who cares about memory hierarchy

The gap between memory storage size and processor performance is widening. So we need to bridge this gap, and making a cache is our strategy.



Processor-Memory Performance Gap "Tax"

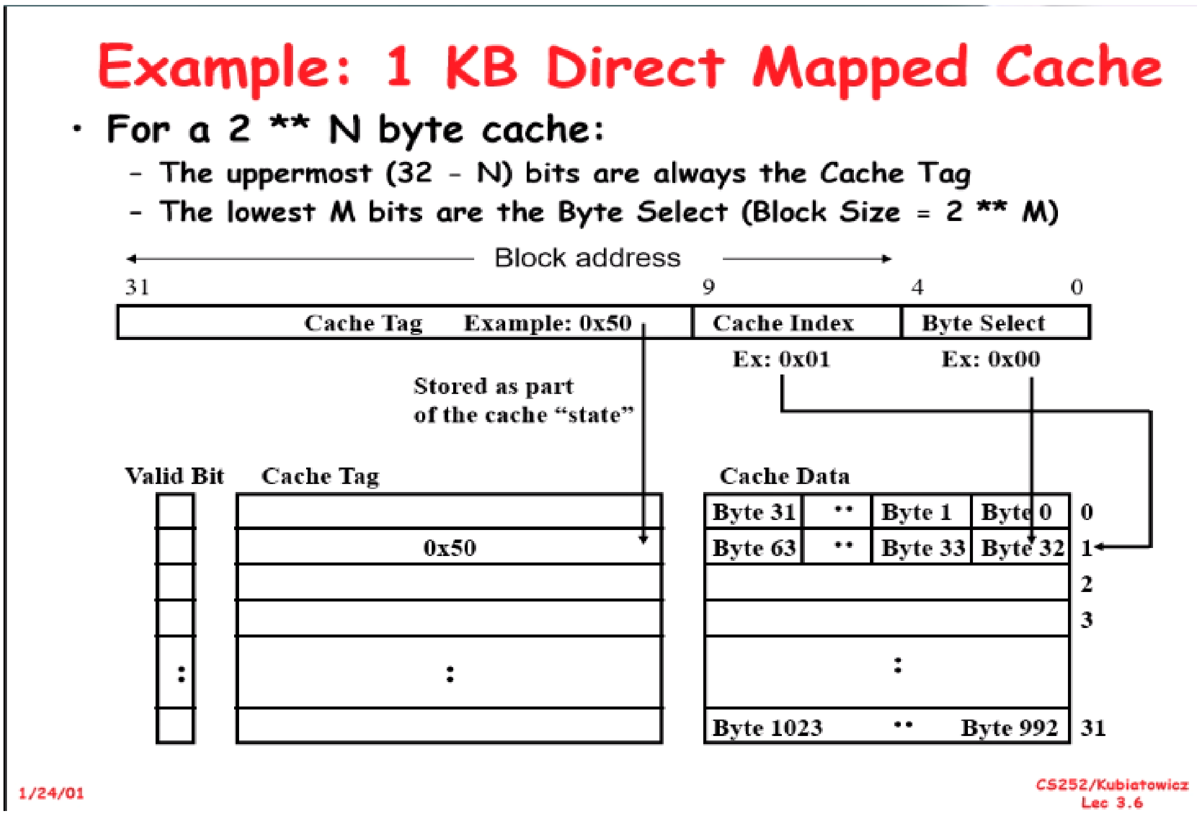
Processor	% Area (-cost)	%Transistors (-power)
• Alpha 21164	37%	77%
• StrongArm SA110	61%	94%
• Pentium Pro	64%	88%

- 2 dies per package: Proc/I$/D$ + L2$

• Caches have no inherent value, only try to close performance gap

2. Cache design examples

2.1 Direct mapped cache



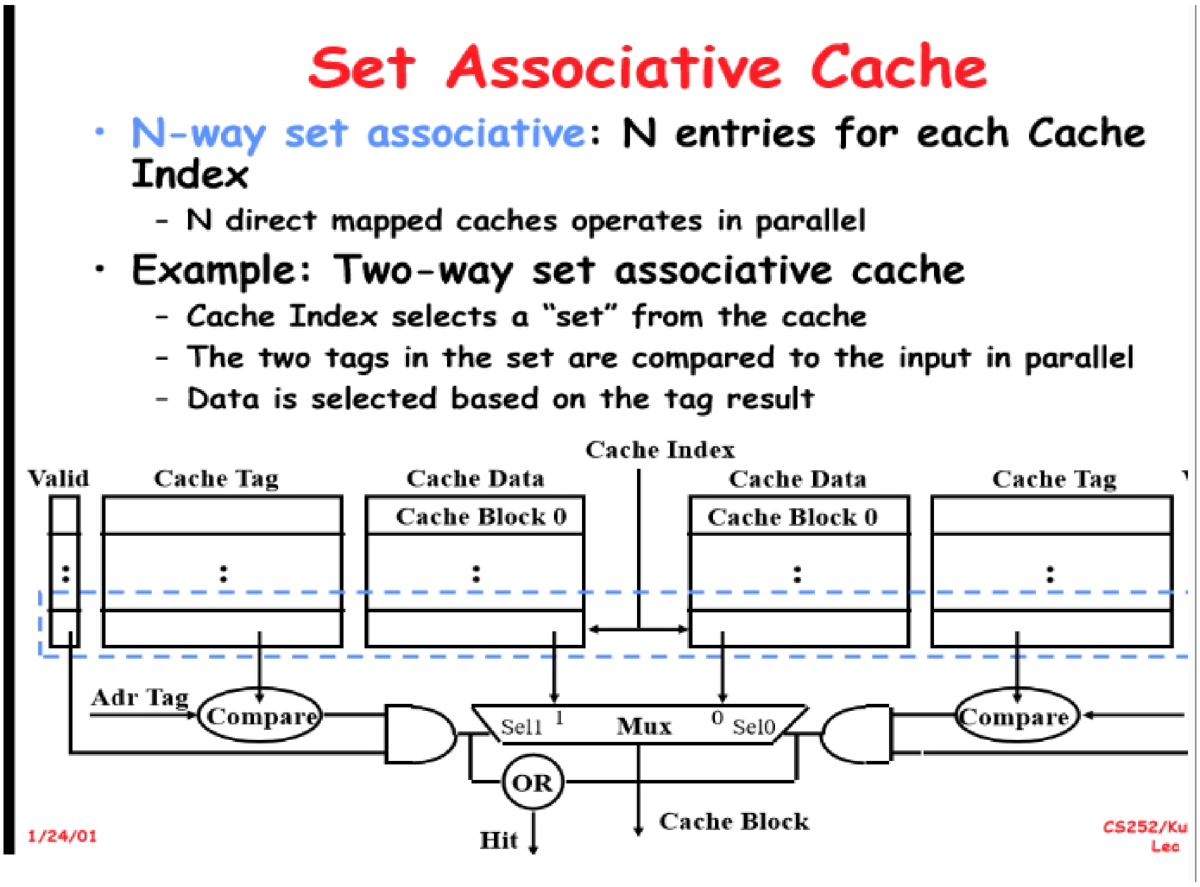
Example: 1 KB Direct Mapped Cache

• 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)

1/24/01

CS252/Kubiatowicz Lec 3.6

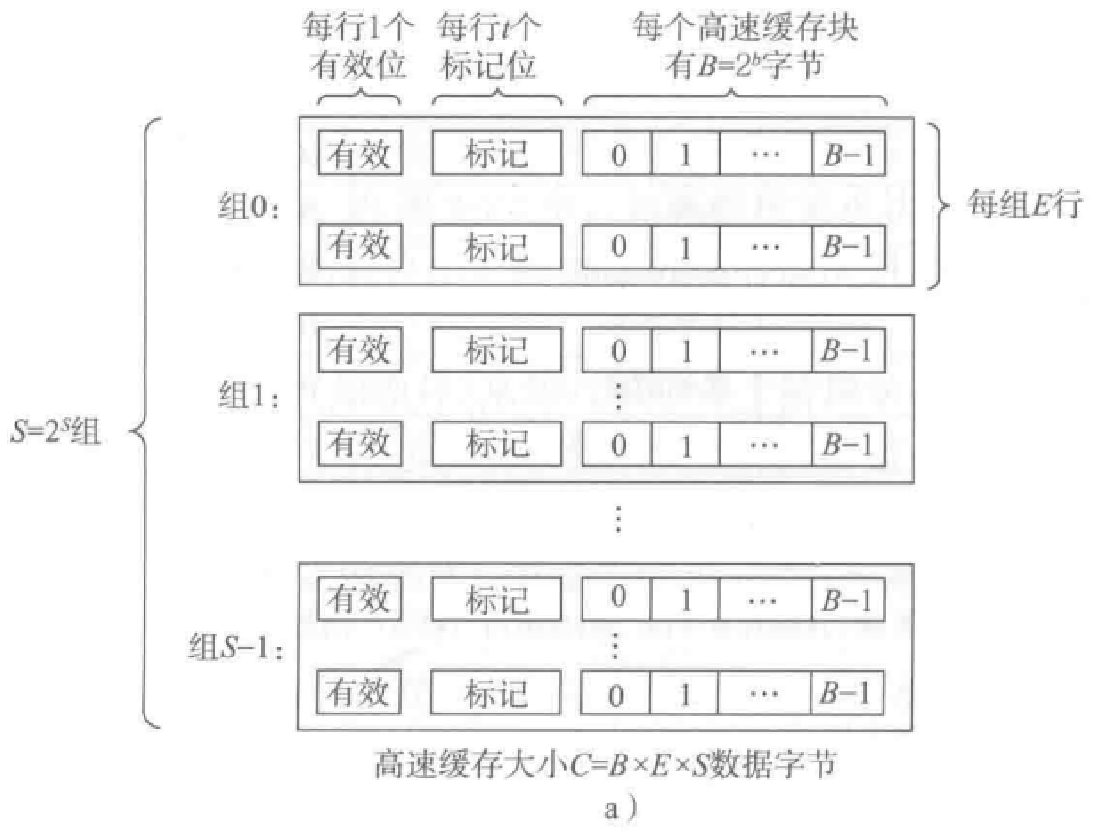
2.2 Set-associative cache

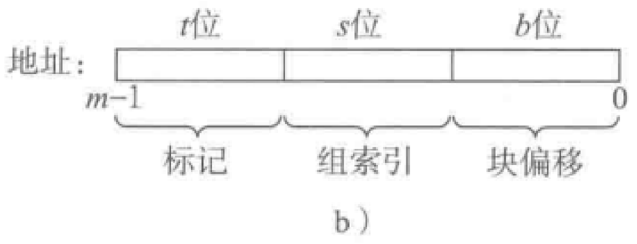


Set Associative Cache

- 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 to the input in parallel
  - Data is selected based on the tag result

Here I would like to use a figure in CSAPP to elaborate:





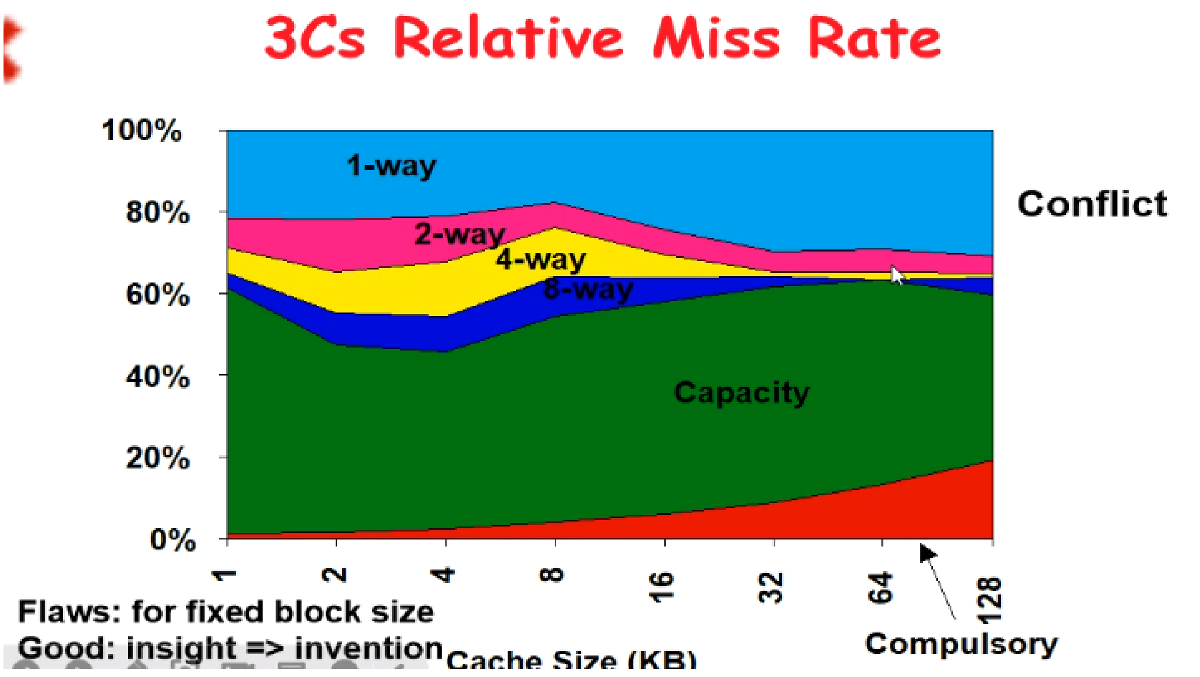
This is the figure of a two-way associated cache. Assume that your address consists of m bytes, and there are 2^s cache groups altogether, each containing E cache lines. Each line contains a data block whose size is $B = 2^b$. Besides, there's a valid bit to signify whether **there IS** data in this block, and there are $t = m - (b + s)$ tag bits, showing the upper bits of the data.

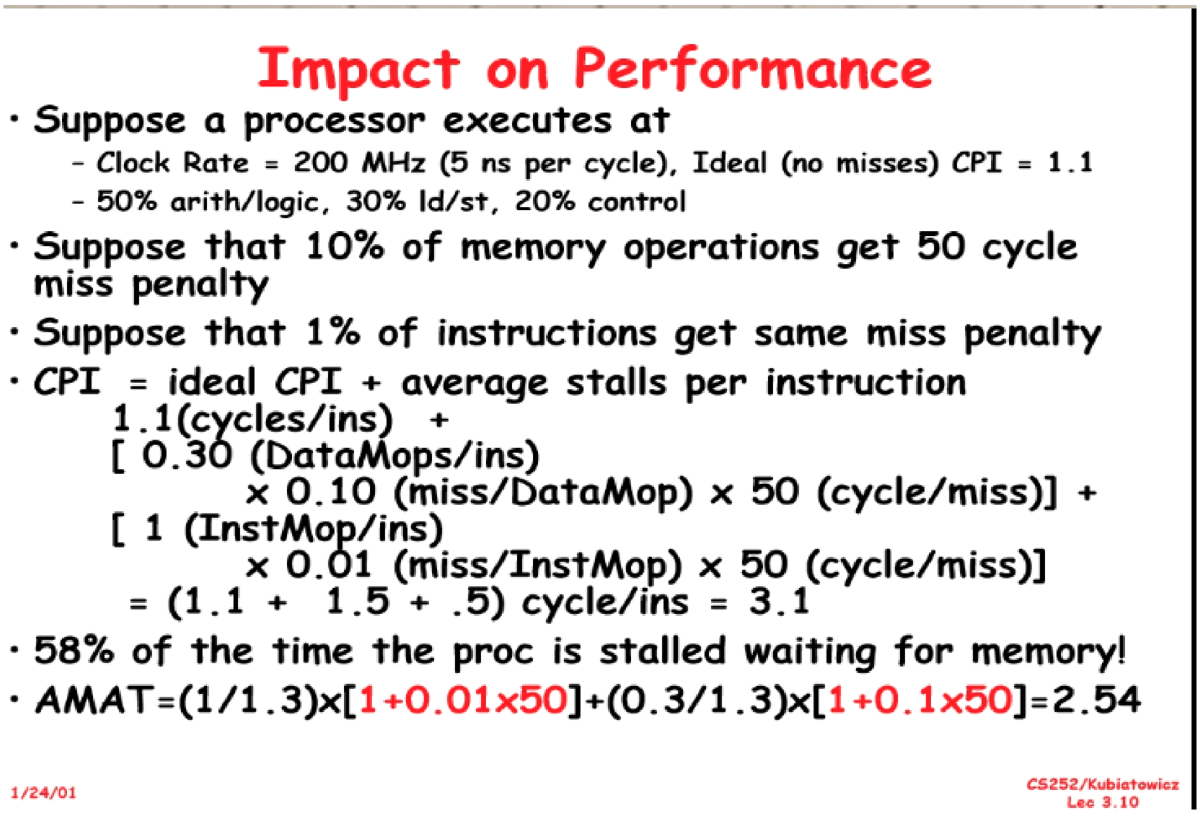
Beware, if this is a two-way associated cache with **LRU substitution strategy**, you need **additional** $\lfloor \log_2 E! \rfloor$ **bits** to store the information about which block is the newest and which block is the oldest.

Obviously, for a N-way cache. If $N = 1$, then this is direct mapped. On the other hand, if $N = N_{max}$, then it's fully associative.

3. Cache performance

3.1 Cache's impact on performance





Impact on Performance

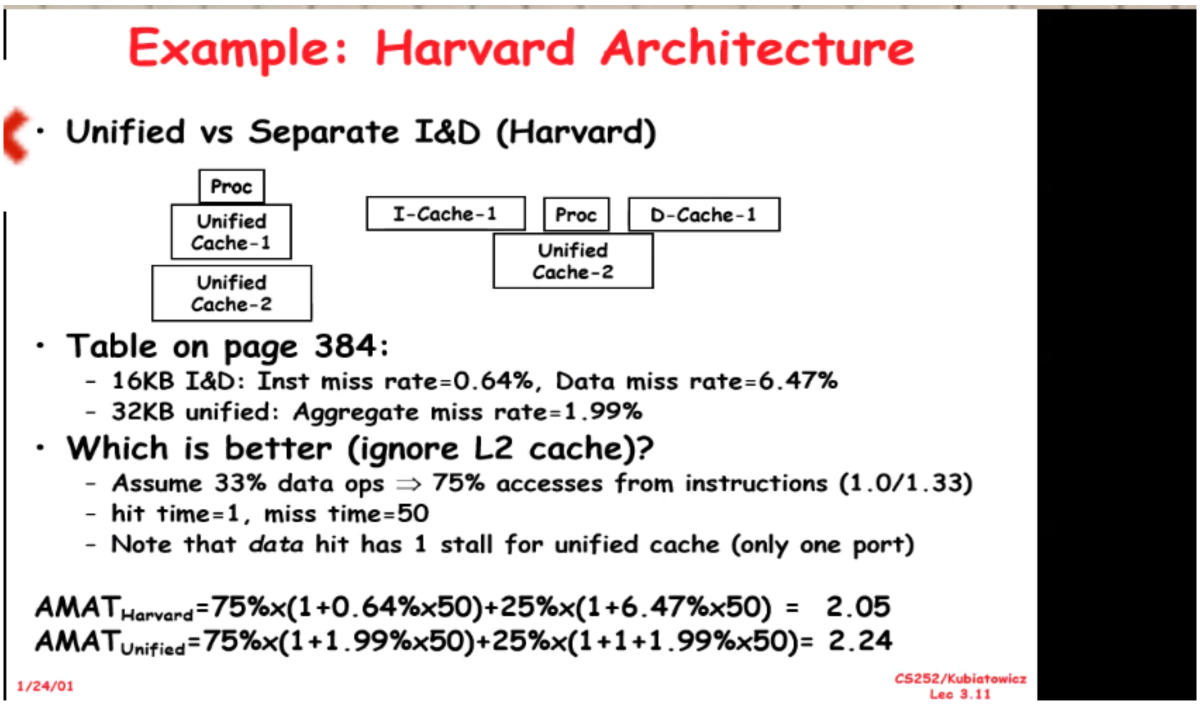
- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- CPI = ideal CPI + average stalls per instruction

  1.1(cycles/ins) +
  [0.30 (DataMops/ins)
  x 0.10 (miss/DataMop) x 50 (cycle/miss)] +
  [1 (InstMop/ins)
  x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1
- 58% of the time the proc is stalled waiting for memory!
- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

1/24/01

CS252/Kubiatowicz
Lec 3.10

3.2 Difference between Harvard architecture and Von-Neumann architecture



Example: Harvard Architecture

- Unified vs Separate I&D (Harvard)

Proc
Unified Cache-1
Unified Cache-2
I-Cache-1
Proc
D-Cache-1
Unified Cache-2

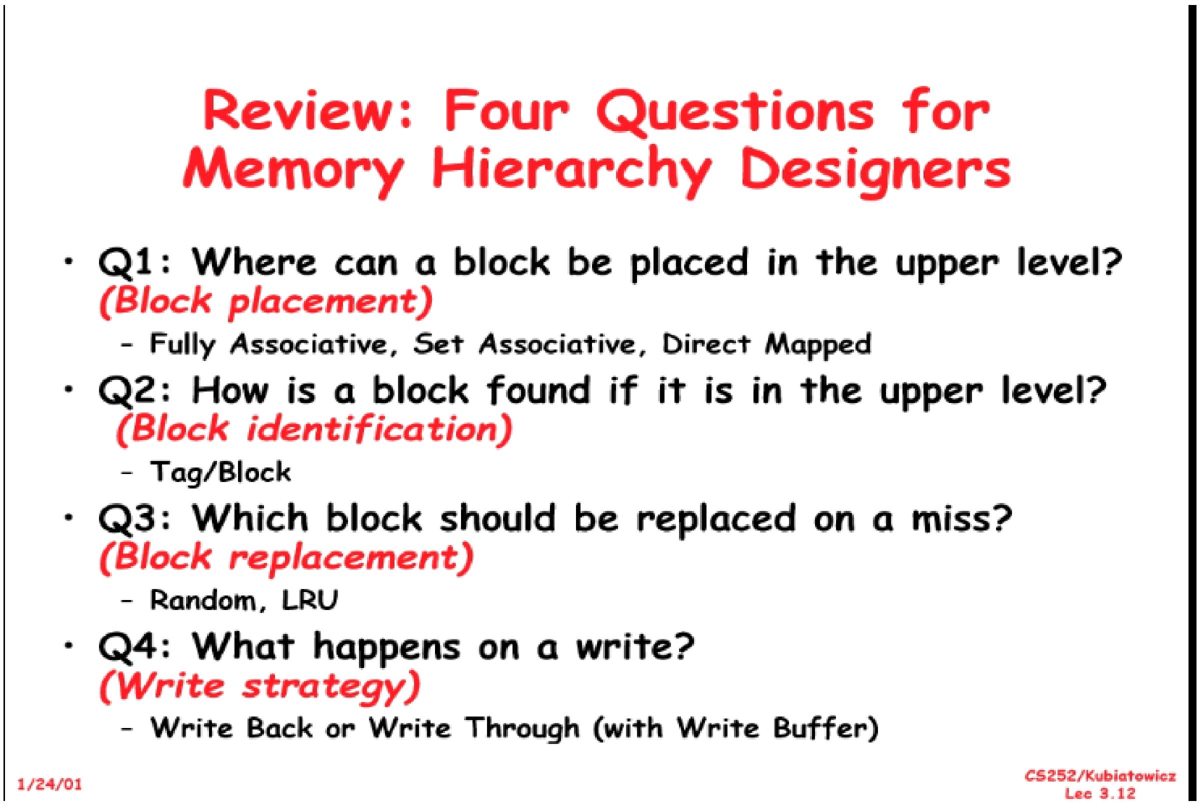
- Table on page 384:
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
- Which is better (ignore L2 cache)?
  - Assume 33% data ops $\Rightarrow$ 75% accesses from instructions (1.0/1.33)
  - hit time=1, miss time=50
  - Note that *data* hit has 1 stall for unified cache (only one port)

$AMAT_{Harvard}$=75%x(1+0.64%x50)+25%x(1+6.47%x50) = 2.05
$AMAT_{Unified}$=75%x(1+1.99%x50)+25%x(1+1+1.99%x50)= 2.24

1/24/01 CS252/Kubiatowicz Lec 3.11

The most significant difference is that Harvard Architecture has separate I-cache and D-cache, which allows it to have less $AMAT$.

4. Summary



Review: Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? *(Block placement)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level? *(Block identification)*
  - Tag/Block
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

1/24/01 CS252/Kubiatowicz Lec 3.12

For specific optimization strategies, please, once again, refer to [this note for Arch revision]().